Comprando esta edição tenha acesso na WEB a:
+280 vídeo-aulas e +140 artigos

www.clubedelphi.net **Feita para desenvolvedores**

ClubeDelphi

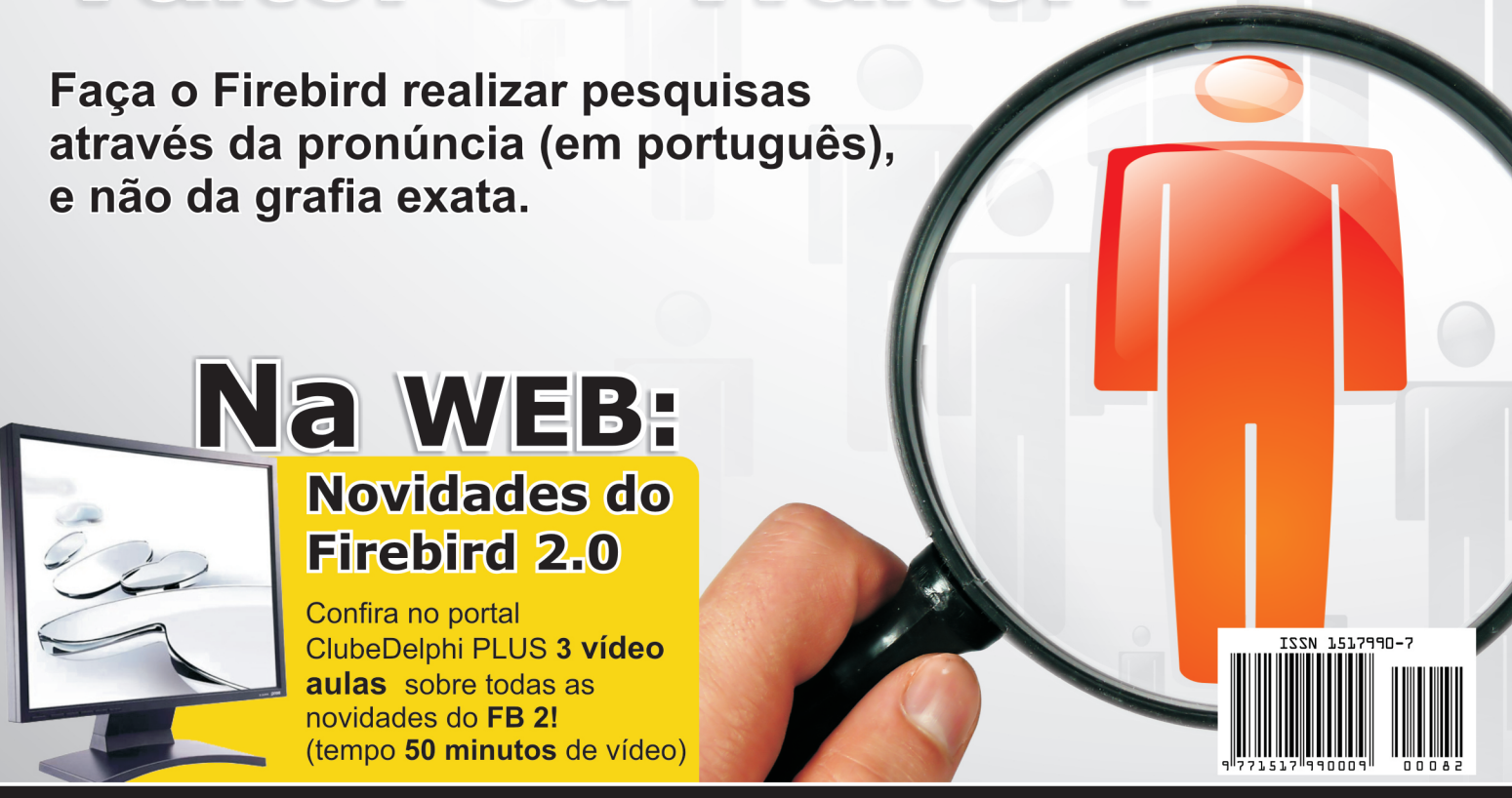
Ano 6 · Edição 82 · R$9,90 DevMedia GROUP

Especial

Crie seu próprio leitor de RSS em Delphi

SQL: Luís ou Luiz? Valter ou Walter?

Faça o Firebird realizar pesquisas através da pronúncia (em português), e não da grafia exata.



Na WEB:

Novidades do Firebird 2.0

Confira no portal ClubeDelphi PLUS **3 vídeo aulas** sobre todas as novidades do **FB 2!** (tempo **50 minutos** de vídeo)

ISSN 1517990-7

CEP dos Correios
Faça sua aplicação VCL Win32 obter dados de CEP a partir de consulta on-line ao Web Service dos correios

Delphi e Outlook
Mala direta com emails personalizados

Coluna: Ask the Expert!

ASP.NET
Crie pesquisas lookup usando janelas estilo "popup"

VCL
Soluções rápidas com o objeto Tapplication

Firebird
Entenda o plano de execução e saiba otimizar consultas e índices

POO
Uma abordagem prática no desenvolvimento de aplicações OO no Delphi - Parte 2

Seu potencial. Nossa inspiração.™

Microsoft®

Visual Studio® 2005.
A diferença é óbvia.

© 2006 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Basic, Visual Studio, o logo Visual Studio e "Seu potencial. Nossa inspiração." são marcas comerciais, registradas ou não, da Microsoft Corporation nos Estados Unidos e/ou outros países. Os nomes de companhias e produtos reais mencionados aqui podem ser marcas comerciais de seus respectivos proprietários.

iu a diferença? Os seus colegas também vão ver. Os atalhos mais apidos do Visual Basic® 2005 permitem que você explore ao máximo s suas habilidades, ao mesmo tempo em que ajudam a solucionar os rojetos mais desafiadores – aqueles que constroem a sua reputação essoal. São mais de 400 features que facilitam a codificação, para ocê concentrar o seu trabalho no que realmente interessa.

eja todas as 400 diferenças em **www.msdn.com.br/visualstudio2005**

Microsoft® Visual Studio® 2005

NÃO PERCA

PORTAL DO ASSINANTE

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:

- Mais de 260 Vídeo Aulas!
- Diversos Mini-Cursos Gratuitos!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:

Login: DVM.PL

Senha: C45LA

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Jorge Luis Bublitz, Leonardo Barbosa Lima Corrêa Pires, João Carlos da Silva, Marcelo Santos Daiber, Marco Antônio P. Araújo, Rodrigo Sendin, Murilo Costa Monteiro Filho, Henrique Gusso Netzka, Dalant Silveira e Michael Benford

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão! Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ClubeDelphi

Ano 6 - 82ª Edição - 2006 - ISSN 1517990-7 **Impresso no Bras**

Editorial

Nesta edição, o Jorge Bublitz mostra como criar uma rotina em Delphi para ser utilizada no Firebird. Usando a UDF criada, agora você será capaz de processar pesquisas fonéticas no Firebird. Por exemplo, ao digitar o nome "Elen", podem ser retornados registros com "Helen", "Ellen" e assim por diante.

O Leonardo Barbosa, Lucas Rezende e Marco Antônio tratam de um importante aspecto no desenvolvimento de aplicações para o Firebird: desempenho. Quem nunca teve problemas de executar consultas pesadas, que levam minutos para retornar? Após ler o artigo sobre otimizações, saiba interpretar planos de execução do banco de dados, e detectar pontos de melhoria, como criação de índices.

Marco Antônio, João Carlos e Marcelo Santos unem em uma só aplicação os principais recursos da POO, incluindo herança, classes, encapsulamento, mapeamento O/R (objeto relacional), persistência e muito mais. Confira a segunda parte do artigo.

Em aplicações desktop normalmente temos aquelas pesquisas auxiliares que chamamos de lookups. Mas como fazer isso em uma aplicação Web, onde não existe o conceito de janelas modais? Veja no artigo do Rodrigo Sendin.

O Murilo Costa mostra como integrar Delphi e Outlook através de Automation, formando um excelente mecanismo de mala direta. O Henrique Netzka apresenta uma interessante técnica para capturar mensagens de erro não tratadas na aplicação cliente.

RSS é um tema em alta atualmente. Quem não tem um leitor de notícias, que faz a busca de novidades nos principais sites de tecnologia, por exemplo? Muito prático, uma vez que não precisamos abrir dezenas de páginas para procurar a informação desejada. E que tal criar um leitor RSS em Delphi? O Michael Benford mostra que isso é possível.

No segundo artigo sobre integração do Delphi com os correios, o Dalan Silveira continua o exemplo que foi apresentado na edição anterior (em .NET), porém agora fazendo uso do Win32 e objetos VCL.

Bom leitura e sucesso com o Delphi.

Guinther Pauli – guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET,
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

ÍNDICE

08 - Código fonético no Firebird
Jorge Luis Bublitz

12 - Otimização de consultas no Firebird
Leonardo Barbosa Lima Corrêa Pires

18 - Uma abordagem prática em aplicações OO no Delphi – Parte 2
João Carlos da Silva, Marcelo Santos Daiber e Marco Antônio P. Araújo

24 - Pesquisas estilo Lookup usando janelas popup na Web
Rodrigo Sendin e Luciano Pimenta

28 - Envio automatizado de emails com Delphi e Outlook
Murilo Costa Monteiro Filho

34 - Pensando simples com a classe TApplication
Henrique Gusso Netzka

38 - Integrando sua aplicação com a busca de CEPS dos Correios – Win32
Dalant Silveira

44 - Crie passo a passo um leitor de RSS com o Delphi
Michael Benford

Ask The Expert

Perguntas e Respostas

Como colorir os itens do RadioGroup e CheckListBox

Gostaria de dar uma incrementada no sistema deixando-o mais colorido e intuitivo, por isso gostaria de colorir os itens do RadioGroup quando marcados.

Juarez Barbosa

Para colorir o item do *RadioGroup* marcado pelo usuário, usamos um recurso bastante simples. É necessário fazermos um *typecast* na classe *RadioGroup* passando pra ela o controle (*Controls*) selecionado e verificando se o mesmo está checado.

Estando checado, trocamos o estilo da fonte para negrito (*fsBold*) e cor azul (*clBlue*). Os demais itens terão a fonte alterada para cor preta (*clBlack*) e nenhum estilo (*[]*). Faremos isso no evento *OnClick* do *RadioGroup* usando um laço *for*, conforme temos na **Listagem 1**.

Listagem 1. Colorindo o item marcado do RadioGroup

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
var
  I: Integer;
begin
  for I := 0 to RadioGroup1.Items.Count - 1 do
    if TRadioButton(RadioGroup1.Controls[
      I]).Checked then
    begin
      TRadioButton(RadioGroup1.Controls[
        I]).Font.Color := clBlue;
      TRadioButton(RadioGroup1.Controls[
        I]).Font.Style := [fsBold];
    end
    else
    begin
      TRadioButton(RadioGroup1.Controls[
        I]).Font.Color := clBlack;
      TRadioButton(RadioGroup1.Controls[
        I]).Font.Style := [];
    end;
end;
```

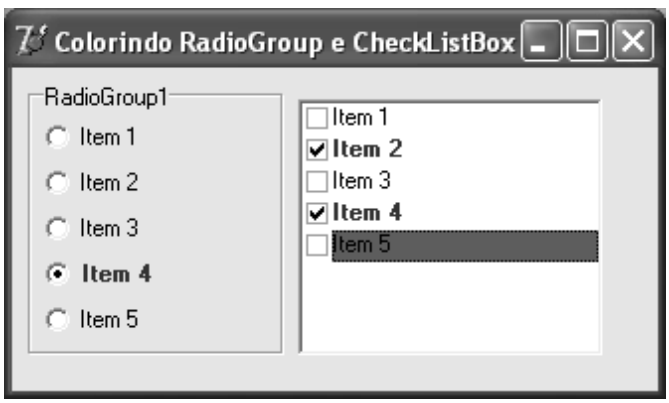


Figura 1. RadioGroup e CheckListBox com itens coloridos

O *CheckListBox* também permite colorir seus itens quando marcados, porém a técnica é diferente e ainda mais simples que no *RadioGroup*. Consiste em "redesenhar" o item selecionado mudando suas propriedades de fonte no evento *OnDrawItem*. Precisamos alterar a propriedade *Style* para *lbOwnerDrawFixed*

Primeiro checamos se o item passado no evento (*Index*) é o item selecionado. Estando selecionado alteramos sua cor para azul (*clBlue*) e o estilo para negrito (*fsBold*), caso contrário alteramos sua cor será preta (*clBlack*) e retiramos o estilo negrito (*[]*).

Após isso chamamos os métodos *FillRect* e *TextOut* que fazem o preenchimento e "redesenho" do texto respectivamente, conforme temos no **Listagem 2**.

Listagem 2. Colorindo o item marcado do CheckListBox

```
procedure TForm1.CheckListBox1DrawItem(
  Control: TWinControl; Index: Integer;
  Rect: TRect; State: TOwnerDrawState);
begin
  with (Control as TCheckListBox) do
  begin
    if Checked[Index] then
    begin
      Canvas.Font.Color := clBlue;
      Canvas.Font.Style := [fsBold];
    end
    else
    begin
      Canvas.Font.Color := clBlack;
      Canvas.Font.Style := [];
    end;
    Canvas.FillRect(Rect);
    Canvas.TextOut(Rect.Left, Rect.Top, (
      Control as TCheckListBox).Items[Index]);
  end;
end;
```

Veja na **Figura 1** os itens selecionados e coloridos.

Adriano Santos

Problemas com IE7 no Delphi 2005/2006

Gostaria de uma orientação para o seguinte problema, que está acontecendo tanto na versão 2005 como na versão 2006 do Delphi. Ao utilizar o editor de código e tentar acessar o editor de design ocorre um erro de Acesso Negado.

Saulus Ribeiro

Olá Saulus, esse problema ocorre depois da instalação da versão 7 do Internet Explorer. Para resolver o problema, acesse o seguinte link que explica passo a passo como resolver esse problema: *www.stevetrefethen.com/blog/ResolvingAccessDeniedErrorsInTheBDSASPNETDesignerWithIE7Installed.aspx*.

Luciano Pimenta

Portal ClubeDelphi PLUS

+260 vídeo aulas

www.clubedelphi.net/portal

Caro Leitor,

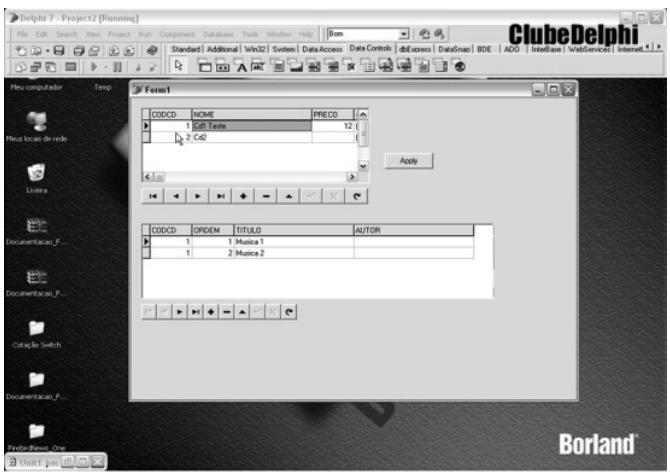
O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 280 vídeo aulas; ii) diversos mini cursos gratuitos; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 140 artigos exclusivos (que não foram publicados na revista)!;

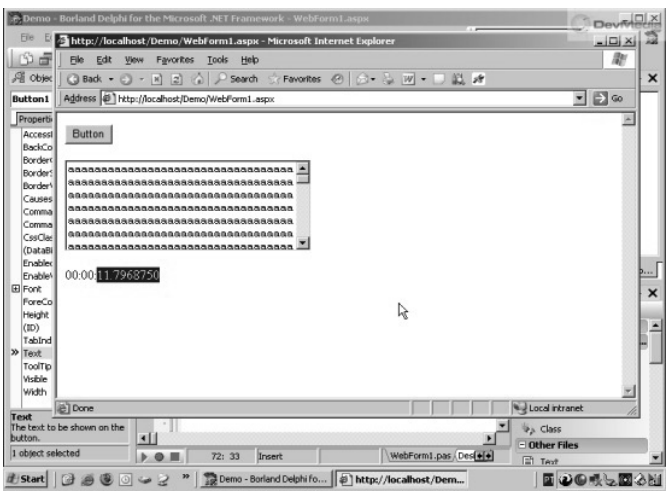
Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!
Equipe DevMedia





Últimas Vídeo-Aulas

Transformações XML

Veja nesta vídeo aula de Guinther Pauli, como realizar transformações XML com o XslTransform do .NET Framework.

Usando a classe StringBuilder

Veja nesta vídeo aula de Guinther Pauli como usar a classe StringBuilder para concatenar strings.

Fazendo backup em XML

Veja nesta vídeo aula de Guinther Pauli, como fazer backup de um banco no formato XML.

Aplicações em camadas

Veja nesta vídeo aula de Guinther Pauli, como criar aplicações em camadas usando o ADO.NET.

.NET Remoting e Firebird

Veja nesta vídeo aula de Guinther Pauli, como construir aplicações multicamadas com .NET Remoting e Firebird.

Lendo documentos XML

Veja nesta vídeo aula de Guinther Pauli, como ler documentos XML com a classe TXMLDocument.

Criando um Template Processor

Veja nessa vídeo aula de Guinther Pauli, como trabalhar com Template Processor através do componente DataSetPageProducer.

Processos do Windows

Veja nessa vídeo aula de Guinther Pauli, como exibir informações sobre Processos do Windows.

Master/Detail com dbExpress

Veja nessa vídeo aula de Fabricio Desbessel como criar a funcionalidade Master/Detail usando dbExpress.

Aplicações Windows Forms no Delphi 2006 - Parte I a IV

Veja nessas aulas de Paulo Quicoli, como trabalhar com aplicações Windows Forms com Delphi 2006.

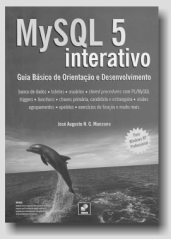
Novidades do Firebird 2.0

Confira 3 vídeo aulas sobre todas as novidades do Firebird 2!

www.devmedia.com.br/articles/listcomp.asp?keyword=fb2videos

Livros de Delphi

Destaque



MySQL 5 Interativo: Guia Básico de Orientação e Desenvolvimento
Érica
5% desconto
De: R$ 89,00
Por: R$ 84,55

Sql Guia Prático 2ª Edição
Brasport

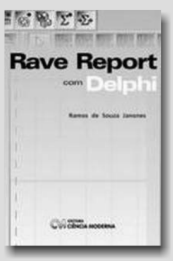
R$ 53,00

Trabalhando com Banco de Dados utilizando o Delphi 2005: Mysql 4, Sql Server 2000, Oracle 9i e Ms-office Access 200
Érica

R$ 97,00

Projeto e Modelagem de Banco de Dados
Campus

R$ 71,76



Rave Report com Delphi
Ciência Moderna

R$ 25,00

Frete Grátis
Para todo o Brasil!

Até 6x sem juros no Visa, Mastercard e Hipercard.
*Parcela mínima de R$ 20,0

Outras facilidades de pagamento:

Tel: (11) 4062-5152

livrosdeprogramacao.com.br

Código fonético no Firebird

JORGE LUIS BUBLITZ
(bublitz@hotmail.com)
é desenvolvedor autodidata desde os 14 anos e formado em Administração de Empresas. Começou com BASIC nos micros de 8 bits, como o Apple II (TK-2000), TRS-80 (CP-500) e MSX (Expert). Desenvolveu em Clipper e em Turbo Pascal desde a versão 3. Desenvolve em Delphi desde a versão 1. Atualmente é funcionário concursado do Tribunal Regional Eleitoral de Mato Grosso e Chefe da Seção de Análise e Desenvolvimento da Secretaria de Tecnologia da Informação, onde coordena e desenvolve diversos projetos cliente/servidor, multicamadas, .NET e VoIP, todos em Delphi.

Um dos maiores problemas em um cadastro, seja de clientes, fornecedores ou produtos, é quando ele se torna muito grande, dificultando a busca por nome ou descrição. Quem nunca se confundiu com o nome de um cliente ou de um produto? Walter ou Valter? Luis ou Luiz? Tem acento? Elizabeth, Elisabette ou Elizabette?

Alguns bancos de dados como o SQL Server e o MySql disponibilizam funções para busca fonética, geralmente chamada *Soundex*, que gera um código fonético de acordo com a pronúncia do idioma inglês, o que a torna impraticável para a língua portuguesa. Até no Clipper tínhamos essa função.

Neste artigo veremos como criar uma rotina de código fonético em português para ser utilizada no Firebird.

Código fonético

Os algoritmos *Soundex* e *Metaphone* são os dois mais utilizados para gerar código fonético. O código *Soundex* tem a propriedade que palavras pronunciadas similarmente produzem a mesma chave *soundex* e assim podem ser usadas em pesquisas em bancos de dados aonde você conhece a pronúncia, mas não exatamente como escreve.

A função retorna uma *string* de quatro caracteres, começando com uma letra, para cada palavra. Assim, por exemplo, para o nome "Pedro da Silva" o código retornado seria algo como *P360D000S410*. Já para "Pedro Silva" o retorno seria algo como *P360S410*.

Já o algoritmo *Metaphone* cria a mesma chave para palavras sonoras similares. Ele é mais preciso do que *Soundex* porque trabalha com as regras básicas da pronunciação. As chaves *Metaphone* geradas são de comprimentos variado. O que faremos é adaptar o algoritmo *Metaphone* para a língua portuguesa.

As regras da nossa função são (quando usarmos o sinal de igualdade significa "tem o som de"):

- Todos os acentos são substituídos, inclusive a cedilha;

• As vogais (“A”, “E”, “I”, “O”, e “U”), o “Y” e o “H” são ignorados;
• “E”, “DA”, “DAS” “DE”, “DI”, “DO” e “DOS” também são ignorados. Ex.: João da Silva = João Silva; Maria do Carmo = Maria Carmo; José Costa e Silva = José Costa Silva;
• Se houver letras duplicadas, a segunda letra é ignorada. Ex: Elizabette = Elizabete;
• As consoantes “B”, “D”, “F”, “J”, “K”, “L”, “M”, “N”, “R”, “T”, “V” e “X” são mantidas;
• A letra “C”:
  o Seguida de “H” tem o som de “X”. Ex.: Chavier = Xavier;
  o Seguida de “A”, “O” ou “U” tem som de “K”. Ex.: Carol = Karol;
  o Seguida de “E” ou “I” tem som de “S”. Ex.: Celina = Selina;
  o Senão é ignorado. Ex: Victor = Vitor;
• A letra “G” seguida de “E” tem som de “J”, senão é mantida. Ex.: Geraldo = Jeraldo;
• A letra “P” seguida de “H” tem som de “F”, senão é mantida. Ex: Phelipe = Felipe;
• A letra “Q” seguida de “U” tem som de “K”, senão é mantida. Ex: Queila = Keila;
• A letra “S”:
  o Seguida de “H” tem o som de “X”. Ex.: Sheila = Xeila;
  o Entre duas vogais tem o som de “Z”. Ex: Casagrande = Cazagrande;
  o Seguido de vogal, é mantido;
  o Senão é ignorado. Ex.: Marcos = Marco;
• A letra “W” tem som de “V”. Ex: Walter = Valter;
• A letra “Z” no final do nome tem som de “S”, senão é mantida. Ex: Luiz = Luis.

Como podemos ver, é uma rotina mais trabalhosa que complexa. A função retorna o código *PDRSLV* tanto para Pedro da Silva como para Pedro Silva.

A DLL

No Delphi crie uma nova DLL (*File>New>Other>DLL Wizard*). Salve tudo em uma nova pasta, dando o nome de “codfon.dpr” para o projeto.

Crie uma nova unit (*File>New Unit*) e salve-a com o nome de “untMain.pas”. Vamos agora digitar o código da **Listagem 1**.

Além disso, no código do projeto recém criado, digite o código a seguir, logo antes do bloco *begin...end*:

```
exports CodiFonPT_BR;
```

Dê um *Build*. Se não houver nenhum erro, será gerada a DLL. Copie o arquivo *codfon.dll* para a pasta *UDF* do Firebird (normalmente em *C:\Arquivos de programas\Firebird\versaoFirebird\UDF*).

E o Kylix?

Você também pode fazer com o Kylix essa rotina. Lembrado que DLL no Linux é SO (Shared Objects), que na prática são a mesma coisa. Tenha cuidado com o nome do projeto, digitando em minúsculas o nome.

Uso da função

Vamos ver como podemos utilizar essa função em um banco já existente. Vamos utilizar o banco de dados do artigo *Uma abordagem prática de Stored Procedures e Triggers*, publicado na edição 77.

No arquivo de download temos o banco *Estoque.fdb* (utilizando Firebird 2.0) com mais de 1800 clientes fictícios cadastrados. Na **Figura 1** temos o diagrama do banco de dados.

Nota: O arquivo para download está na versão 2.0 do Firebird, mas você pode criar a função fonética e usá-la em qualquer versão do servidor.

Para utilizar a função, devemos executar o seguinte comando SQL:

```
DECLARE EXTERNAL FUNCTION FCodiFonPT_BR
CSTRING(255) NULL
RETURNS CSTRING(255) FREE_IT
ENTRY_POINT 'CodiFonPT_BR' MODULE_NAME 'codfon';
```

O código anterior deve ser utilizado no Firebird 2.0. Se você estiver usando uma versão anterior, basta remover o NULL do código. Trabalharemos com a tabela *Cliente*, que possui uma *Trigger* para o auto-incremento do campo *Cod_Cliente*.

O que faremos é adicionar um campo na tabela (*Cod_Fon_Nome*) do tipo *VarChar*, que armazenará o código fonético do nome do cliente. Criaremos também um índice para esse campo, para que as consultas sejam otimizadas pelo banco.

Caso deseje utilizar a DLL no seu banco, poderia tentar simplesmente alterar os comandos SQL de pesquisa, como por exemplo:

```
SELECT COD_CLIENTE, NOME_CLIENTE
FROM CLIENTE
WHERE NOME_CLIENTE = :PNOME_CLI
```

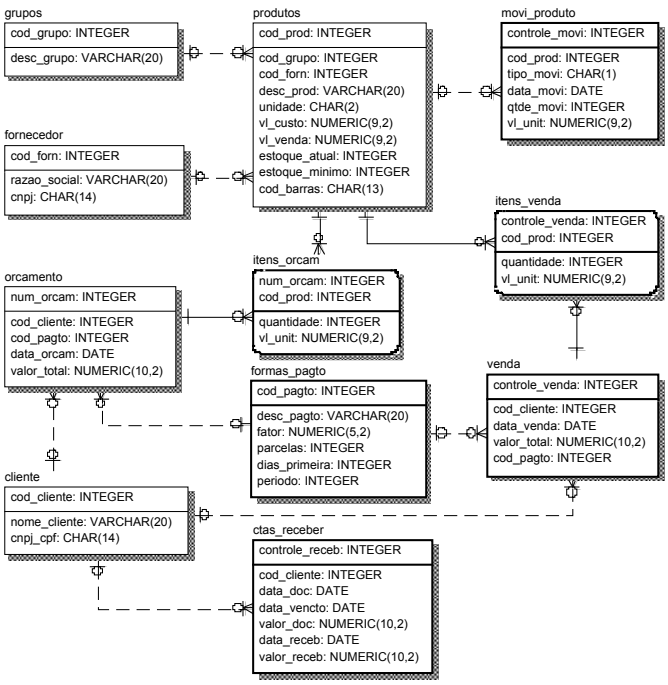


Figura 1. Modelo do banco de exemplo

Troca-se por:

```
SELECT COD_CLIENTE, NOME_CLIENTE
FROM CLIENTE
WHERE FCodiFonPT_BR(NOME_CLIENTE) = FCodiFonPT_BR(:PNOME_CLI)
```

Neste exemplo o banco terá que percorrer toda a tabela, chamando a função para gerar o código fonético em cada registro, para então fazer a comparação. Se a tabela for grande, haverá uma degradação no desempenho. Para adicionar um novo campo na tabela *Cliente*, o comando SQL é:

```
ALTER TABLE CLIENTE
ADD COD_FON_NOME VARCHAR(50)
```

Para criar o índice, o comando é:

```
CREATE INDEX CLIENTE_COD_FON ON CLIENTE (COD_FON_NOME)
```

Precisamos também atualizar o conteúdo desse novo campo:

```
UPDATE CLIENTE
SET COD_FON_NOME = FCodiFonPT_BR(NOME_CLIENTE);
```

E para que o campo seja atualizado automaticamente, seja em inclusão ou alteração, vamos criar uma *Trigger* para fazer isso:

```
CREATE TRIGGER ATU_COD_FON FOR CLIENTE
ACTIVE BEFORE INSERT OR UPDATE POSITION 1 AS
BEGIN
  NEW.COD_FON_NOME = FCodiFonPT_BR(NEW.NOME_CLIENTE);
END
```

Listagem 1. A unit untMain

```
unit untMain;
interface
uses
  SysUtils;

function CodiFonPT_BR(nome: PChar): PChar; cdecl; export;

implementation
function CodiFonPT_BR(nome: PChar): PChar;
var
  i, p: integer;
  novo, aux: string;
begin
  try
    aux := AnsiUpperCase(nome);
    novo := '';
    { Tira acentos e cedilha }
    for i := 1 to Length(aux) do
    begin
      case aux[i] of
        'Á', 'Â', 'Ã', 'À', 'Ä', 'Å': aux[i] := 'A';
        'É', 'Ê', 'È', 'Ë': aux[i] := 'E';
        'Í', 'Î', 'Ì', 'Ï': aux[i] := 'I';
        'Ó', 'Ô', 'Õ', 'Ò', 'Ö': aux[i] := 'O';
        'Ú', 'Û', 'Ù', 'Ü': aux[i] := 'U';
        'Ç': aux[i] := 'C';
        'Ñ': aux[i] := 'N';
        'Ý', 'Ÿ', 'Y': aux[i] := 'I';
      else
        if Ord(aux[i]) > 127 then
          aux[i] := #32;
      end;
    end;
    { Retira E, DA, DAS, DE, DI, DO e DOS do nome
      José da Silva = José Silva
      João Costa e Silva = João Costa Silva }
    p := Pos(' DA ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DA ', aux);
    end;
    p := Pos(' DAS ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 4);
      p := Pos(' DAS ', aux);
    end;
    p := Pos(' DE ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DE ', aux);
    end;
    p := Pos(' DI ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DI ', aux);
    end;
    p := Pos(' DO ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 3);
      p := Pos(' DO ', aux);
    end;
    p := Pos(' DOS ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 4);
      p := Pos(' DOS ', aux);
    end;
    p := Pos(' E ', aux);
    while p > 0 do
    begin
      Delete(aux, p, 2);
      p := Pos(' E ', aux);
    end;
    { Retira letras duplicadas
      Elizabette = Elizabete }
    for i := 1 to Length(aux)-1 do
      if aux[i] = aux[i+1] then
        Delete(aux, i, 1);
    for i := 1 to Length(aux) do
    begin
      case aux[i] of
        { 'A','E','I','O','U','Y','H' e espaços
          ignora }
        'B','D','F','J','K','L','M','N','R','T',
        'V','X':
          novo := novo + aux[i];
        'C':      { CH = X }
          if aux[i+1] = 'H' then
            novo := novo + 'X'
          else   { Carol = Karol }
          if aux[i+1] in ['A','O','U'] then
            novo := novo + 'K'
          else   { Celina = Selina, Cintia = Sintia }
          if aux[i+1] in ['E','I'] then
            novo := novo + 'S';
        'G': { Jeferson = Geferson }
          if aux[i+i] = 'E' then
            novo := novo + 'J'
          else
            novo := novo + 'G';
        'P': { Phelipe = Felipe }
          if aux[i+1] = 'H' then
            novo := novo + 'F'
          else
            novo := novo + 'P';
        'Q': { Keila = Queila }
          if aux[i+1] = 'U' then
            novo := novo + 'K'
          else
            novo := novo + 'Q';
        'S':
          case aux[i+1] of
            'H': { SH = X }
              novo := novo + 'X';
            'A','E','I','O','U':
              if aux[i-1] in ['A','E','I','O','U'] then
                { S entre duas vogais = Z }
                novo := novo + 'Z'
              else
                novo := novo + 'S';
          end;
        'W': { Walter = Valter }
          novo := novo + 'V';
        { no final do nome tem som de S ->
          Luiz = Luis }
        'Z':
          if (i = Length(aux)) or (aux[i+1] = ' ') then
            novo := novo + 'S'
          else
            novo := novo + 'Z';
      end;
    end;
    novo := novo + ' ';
    CodiFonPT_BR := PChar(novo);
  except
    CodiFonPT_BR := PChar('');
  end;
end;
end.
```

Veja que definimos a Posição 1 (*Position 1*) para a *Trigger*. Isso porque na posição 0 temos a *Trigger* que atualiza o campo *Cod_Cliente* (auto-incremento). Veja agora a busca funcionando: vamos procurar todos os clientes com o nome iniciado por Ana Paula, use o seguinte comando SQL:

```
SELECT COD_CLIENTE, NOME_CLIENTE, COD_FON_NOME
FROM CLIENTE
WHERE COD_FON_NOME LIKE
FCodiFonPT_BR('ana paula')||'%'
ORDER BY 2
```

Veja o resultado na **Figura 2**. Veja que não importa se é Ana ou Anna, se é Paula, Paola ou Paulla.

O uso prático

Crie uma nova aplicação no Delphi, de acordo com a **Figura 3**.

Temos um *DBGrid* ("dgCliente"), um *Edit* ("edtNome Cliente"), um *Label* e um *BitBtn* ("btnPesquisar"). Para acessar o banco, um *SQLConnection* ("conFB2"), o trio *SQLDataSet* ("sqlCliente") + *DataSetProvider* ("dspCliente") + *ClientDataSet* ("cdsCliente") e um *DataSource* ("dsCliente").

Configure o *SQLConnection* para acessar o banco que alteramos anteriormente. No *SQLDataSet* aponte a propriedade *SQLConnection* para o *conFB2* e o *CommandText* para:

```
SELECT COD_CLIENTE, NOME_CLIENTE, COD_FON_NOME
FROM CLIENTE where COD_FON_NOME LIKE
FCodiFonPT_BR(:pnome)||'%'
ORDER BY 2
```

No *DataSetProvider* aponte a propriedade *DataSet* para o *sqlCliente*. No *ClientDataSet* aponte a propriedade *Provider Name* para o *dspProvider* e a propriedade *Active* para *True*. No *DataSource* aponte a propriedade *DataSet* para o *cdsCliente*. No *DBGrid* aponte a propriedade *DataSource* para o *dsCliente*. No evento *OnClick* do *btnPesquisar* digite o seguinte código:

```
cdsCliente.Close;
cdsCliente.Params[0].AsString := edtNomeCliente.Text;
cdsCliente.Open;
```

Execute a aplicação. Digite um nome qualquer e clique em *Pesquisar*. Veja o resultado na **Figura 4**.

Conclusão

Neste artigo aprendemos o uso prático de UDF no Firebird, além de aprendermos um pouco sobre os conceitos de códigos fonéticos. Espero que essa rotina seja muito útil a vocês. Abraços e boas compilações. ■

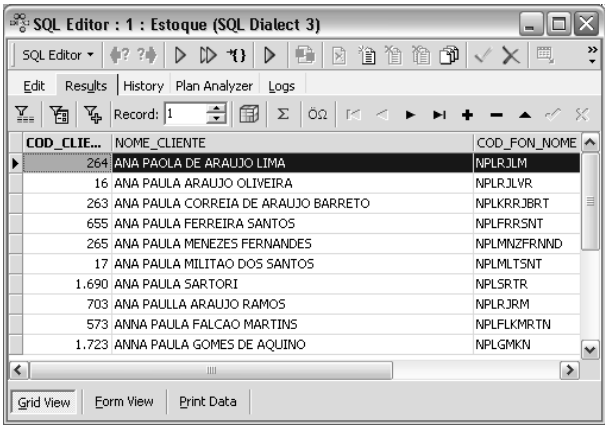


Figura 2. Pesquisa utilizando código fonético

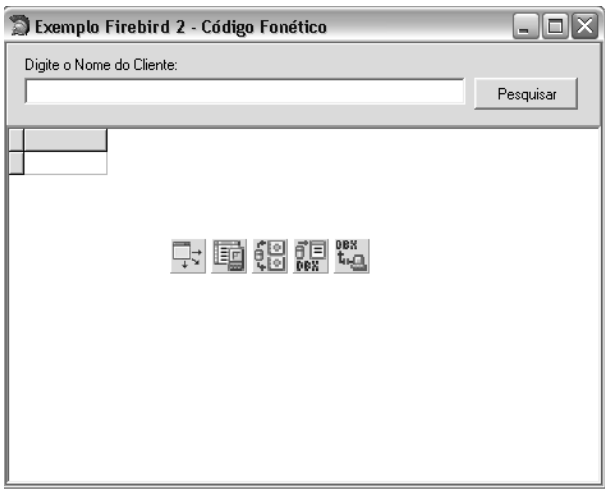


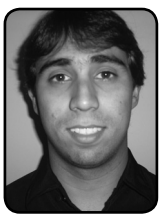
Figura 3. A aplicação de exemplo

Figura 4. Buscando Ana Paula na aplicação de exemplo

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

Otimização de consultas no Firebird

LEONARDO B. L. C. PIRES
(lbpires@gmail.com)
é graduando do Centro de Ensino Superior de Juiz de Fora - CES/JF no curso de Bacharelado em Sistemas de Informação, é um dos membros fundadores do JUGJF (Java User Group de Juiz de Fora) e atualmente vem trabalhando no desenvolvimento de soluções para dispositivos móveis e automação comercial na Hand Mobile Solutions (www.hand.inf.br).



LUCAS REZENDE RICARDO
(lrezender@yahoo.com.br)
é graduando do Curso de Bacharelado de Sistemas de Informação no Centro de Ensino Superior de Juiz de Fora, atua como programador e membro da equipe de administração de dados da Zeus Rio Solutions (www.zeusrio.com.br), é um dos membros fundadores do JUGJF (Java User Group de Juiz de Fora), trabalhando com as ferramentas de desenvolvimento da Sybase, PowerBuilder e tecnologia Microsoft, incluindo banco de dados SQL Server.

MARCO ANTÔNIO P. ARAÚJO
(maraujo@cesjf.br)
é Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora, Analista de Sistemas da Prefeitura de Juiz de Fora.

Independente do tipo de aplicação desenvolvida, Web ou desktop, garantir o seu desempenho é fundamental. Para isso, é preciso analisar e entender os fatores que contribuem para degradação do desempenho de um sistema. Dentre esses fatores, um dos principais refere-se ao desempenho das operações executadas no banco de dados, principalmente das consultas, utilizadas com grande frequência.

Este artigo busca esclarecer algumas dessas questões relativas ao desempenho de consultas em bancos de dados e apresentar as ferramentas disponíveis no Firebird que auxiliam na tarefa de otimização, possibilitando a criação de consultas mais refinadas, objetivando o ganho de desempenho.

Plano de execução

Quando uma operação é submetida ao banco de dados, seja ela um *Select*, *Insert* ou outra qualquer, o Firebird, assim como outros SGBDs, possui um mecanismo interno chamado otimizador de consultas, que tem a tarefa de analisar a operação a ser executada e avaliar a utilização de índices, *Unions*, *Join*s e outros, formando então um plano de execução, chamado de *Plan* no Firebird.

Como resultado do plano de execução tem-se o custo da operação, que é uma forma de avaliar se a consulta obteve um ganho de desempenho ou não. Internamente o otimizador executa uma seqüência de passos para escolher qual tipo de leitura será utilizado, se serão feitos diretamente sobre os registros ou através de estruturas de índices, bem como informar o tempo utilizado e o total de registros percorridos.

Cada consulta executada possui um plano específico que pode ser definido automaticamente pelo otimizador do Firebird ou manualmente pelo desenvolvedor da aplicação. Se esse otimizador avaliar que o custo da operação foi ruim, ele pode submeter a consulta a outra otimização ou então não usar otimização, utilizando um *Natural Plan*.

Configurando um banco de dados para estudo

Para os estudos de otimização será utilizado o banco *Employee.fdb* que acompanha o pacote de instalação do Firebird

e encontra-se na pasta *examples* no diretório de instalação do banco de dados.

Como ferramenta para administração será utilizado o IBExpert que é uma ferramenta comercial para administração de bancos de dados da família InterBase e Firebird, mas possui uma versão funcional *free*, obtida no endereço *www.ibexpert.com*.

Uma vez que o banco de dados já está criado, será necessário apenas registrá-lo no IBExpert para que seja possível executar as tarefas de administração. Os passos a seguir descrevem como registrar o banco no IBExpert.

Ao acessar o menu *Database>Register Database* um assistente como o da **Figura 1** será aberto. É nele que serão realizadas as configurações de registro do banco.

Em *Server* é preciso informar onde está instalado o servidor de banco de dados e possui duas opções: *Local*, no caso do servidor de banco de dados estar instalado na própria máquina que está trabalhando, e *Remote* para o caso de conectar a uma máquina servidora remota.

Quando escolher *Remote*, é preciso informar outros dois parâmetros que são *Server Name*, ou seja, o endereço da máquina servidora e *Protocol*, que é o protocolo para estabelecer a conexão com a máquina servidora. No caso deste exemplo, o servidor de banco de dados está situado na máquina local.

Quanto aos outros campos, *Database* e *Client Library File*, o primeiro refere-se ao caminho do arquivo de banco de dados que será registrado e o segundo representa a biblioteca utilizada para conexão com o banco.

O usuário administrador do banco de dados é o padrão, já criado quando é feita a instalação do Firebird, ou seja, o SYSDBA e a sua senha é "masterkey". Depois de preenchidos os parâmetros de registro é preciso clicar em *Register* para registrar o banco de dados.

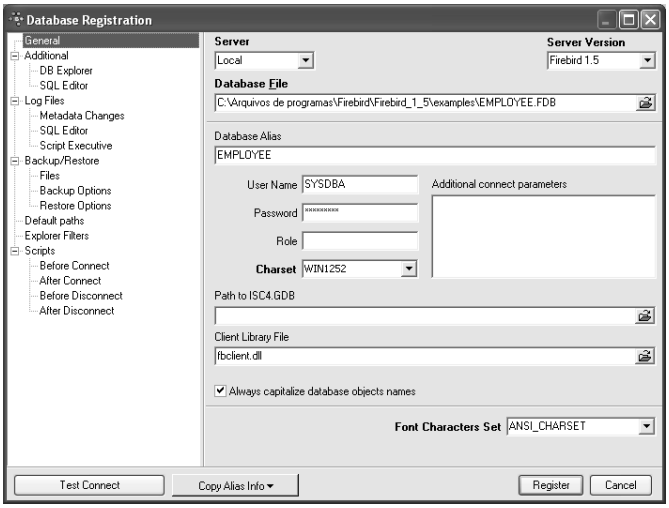


Figura 1. Assistente de registro de bancos de dados

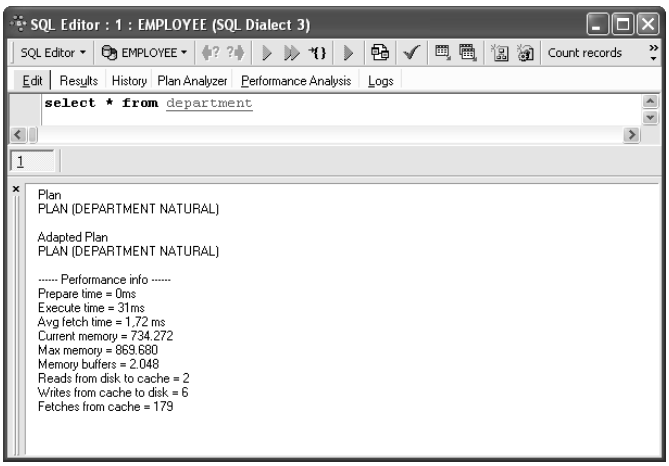


Figura 2. Janela do SQL Editor

Entendendo o Plano de Execução

Para um primeiro contato com o plano de execução do Firebird será executada uma consulta simples buscando listar todos os departamentos da tabela *Department* utilizando a ferramenta de execução de consultas do IBExpert, o *SQL Editor*.

O acesso ao *SQL Editor* é feito através do menu *Tools>SQL Editor*. Com editor aberto, basta digitar o comando SQL: "select * from department" e clicar no botão *Execute*, indicado por uma seta verde ou apertar a tecla de atalho F9 para executar a consulta (**Figura 2**).

A janela do *SQL Editor* é composta por diversas abas, sendo que, para os estudos de otimização, as mais utilizadas serão *Edit*, onde serão digitados os comandos SQL, *Plan Analyzer*, que informa em detalhes o plano de execução criado e executado para a consulta e *Performance Analysis*, responsável por exibir graficamente os detalhes do desempenho da consulta executada.

Na aba *Plan Analyzer* é possível visualizar com detalhes o *Plan* gerado para a consulta e os tipos de acesso que foram utilizados para fazer a leitura das tabelas envolvidas na consulta que, nesse caso, foi *Natural*, ou seja, uma leitura seqüencial sem a utilização de índices (**Figura 3**).

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como instalar e trabalhar com o IBExpert.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Na aba *Performance Analysis* é possível visualizar graficamente o desempenho da consulta. Nessa consulta o gráfico está representando uma leitura não indexada, ou seja, nenhum índice foi utilizado para leitura da tabela e, no final da barra, o total de registros percorridos é apresentado que, nesse caso, são 21 (**Figura 4**).

Na parte inferior da janela do *SQL Editor* encontra-se a aba *Messages* onde são apresentadas as estatísticas da consulta que foi executada e que são detalhadas a seguir:

• *Prepare time*: Tempo requerido para preparar a consulta;
• *Execute time*: Tempo de execução da consulta;
• *Avg fecht time*: Tempo médio de busca;
• *Current memory*: Memória utilizada;
• *Max memory*: Quantidade máxima de memória disponível por operação;
• *Memory buffers*: Tamanho do *buffer* de memória para armazenamento;
• *Reads from disk to cache*: Leituras feitas no disco e colocadas na memória;
• *Writes from cache to disk*: Escritas executadas em cache;
• *Fetches from cache*: Buscas no cache de disco.

Os dados apresentados em *Performance Info* são indicadores de desempenho, com destaque para o *Execute Time* que é o tempo de execução da consulta. Quando o *Execute Time* é elevado pode significar que o desempenho da consulta está comprometido.

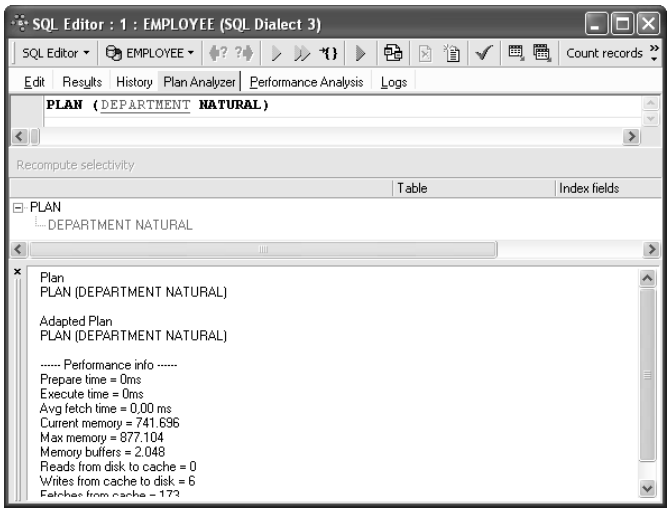


Figura 3. Aba Plan Analyzer

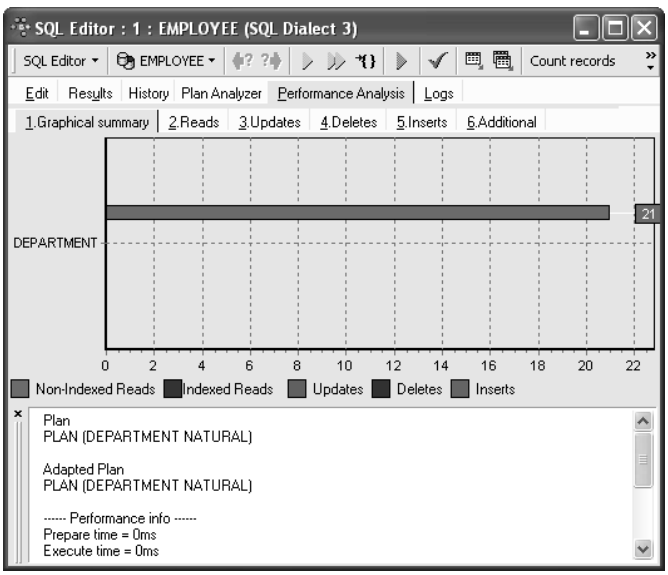


Figura 4. Aba Performance Analysis

Para todo *Plan* gerado pelo otimizador do Firebird a sintaxe básica será a palavra *Plan* acompanhada de expressões que identificará o tipo de acesso realizado ao ler as tabelas, que tabelas foram acessadas, se houve *Join* entre tabelas e a ordem de manipulação das mesmas.

Pegando como exemplo o plano gerado para listar todos os departamentos da tabela, a consulta e o plano gerado podem ser vistos na **Figura 2**. Nesse plano é possível observar duas expressões, além do comando *Plan*. A primeira é a tabela que está sendo lida, *Department*, e a segunda é o tipo de leitura na tabela, no caso *Natural*.

Quando temos a expressão *Natural* significa que a leitura foi feita de forma seqüencial, sem utilização de índices, ou seja, sem otimização. Em alguns casos não otimizar poderá ser uma melhor opção, uma vez que o custo para acessar o índice poderá ser maior do que uma leitura seqüencial, ou até mesmo devido à necessidade da consulta de acessar todos os registros mesmo que existam índices.

Utilizando índices

Internamente, o banco de dados é armazenado em blocos de disco e, a cada consulta, as páginas que armazenam dados da tabela são varridas. Nessas páginas estão todas as linhas que o Firebird lerá para retornar o resultado. Essa varredura pelas páginas pode ser feita de maneira seqüencial, onde todas as linhas são lidas ou com a utilização de índices.

Os índices são usados somente quando são bastante seletivos, ou seja, o resultado trazido deve possuir no máximo 5% do total de linhas armazenadas. Toda essa análise é feita internamente pelo SGBD para que obtenha um ganho no custo estimado pela execução do comando.

Quando uma consulta é submetida, o otimizador do Firebird sempre verifica se existem índices que poderão ser utilizados com o intuito de reduzir o tempo de execução da consulta e, por sua vez, o seu custo. Basicamente o Firebird busca fazer uso de índices para executar três tipos de tarefas diferentes, são elas: comparações de igualdade, *Joins* entre tabelas e cláusulas de *Order By* e *Group By*.

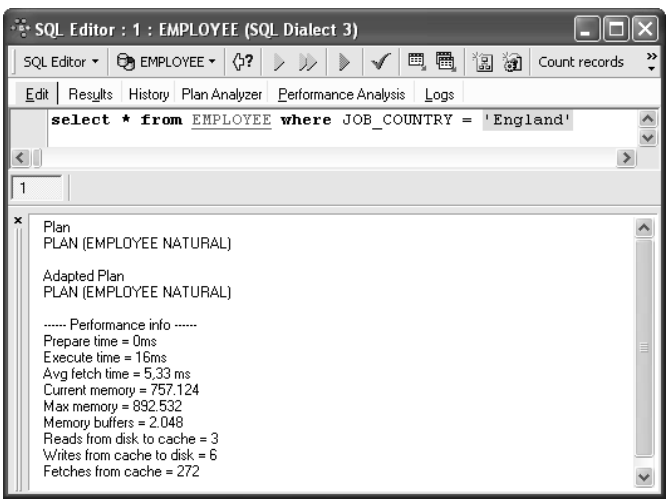


Figura 5. Consulta com cláusula where de comparação

Nas consultas que utilizam cláusulas de "maior que", "menor que" ou "igual a", ou seja, comparações de igualdade, os índices serão utilizados para remover registros que não cumprem a condição estipulada, evitando assim a leitura desnecessária desses.

Para exemplificar, foi elaborada uma consulta que utiliza da cláusula de comparação de igualdade *where* que lista todos os empregados da tabela *Employee* que trabalham na Inglaterra (*England*), com o seguinte código:

```
select * from EMPLOYEE
where JOB_COUNTRY = 'England'
```

Veja o resultado na **Figura 5**.

O plano gerado para essa consulta foi uma leitura seqüencial na tabela *Employee*, comparando quais registros cumprem a cláusula de filtro *where*. Como visto anteriormente, quando o tipo de acesso é *Natural* em uma tabela, significa que todos os registros foram lidos e comparados, o que certamente demanda uma quantidade de tempo maior do que acessar uma estrutura de índices para buscar os registros que satisfazem à condição.

Agora será criado um índice para a coluna utilizada na cláusula *where* que é a *Job_Country*, com o seguinte código:

```
Create Index IDX_EMPLOYEE_JOB_COUNTRY on Employee (JOB_COUNTRY)
```

Para efetuar a criação do índice, é necessário executar o código anterior no *SQL Editor* e em seguida, executar um *Commit* na transação pressionando as teclas CTRL+ALT+C. Após criar o índice e executar novamente a consulta, é possível observar que o plano de execução mudou (**Figura 6**).

O tipo de acesso passou para *Index* e o tempo de execução diminuiu. A diminuição do tempo de execução é ainda afetada por diversos fatores, como a capacidade de processamento do servidor, o número de tabelas envolvidas, a quantidade de registros em cada tabela e o tamanho do registro.

Outra ocasião que faz com que o otimizador do Firebird opte pela utilização de índices é quando há a necessidade de realizar um *Join* entre tabelas, nesse caso os índices poderão ser utilizados pelo otimizador para realizar a operação com mais eficiência.

Quando existe pelo menos um índice em uma das colunas utilizadas na cláusula de *Join*, a coluna que não possui índices será lida primeira e os valores a serem encontrados serão procurados na estrutura de índices da outra coluna.

Para compreender melhor a questão da utilização de índices nos *Joins*, o índice para a coluna *Job_Country* que foi criado no exemplo anterior será removido com o seguinte código:

```
DROP INDEX IDX_EMPLOYEE_JOB_COUNTRY
```

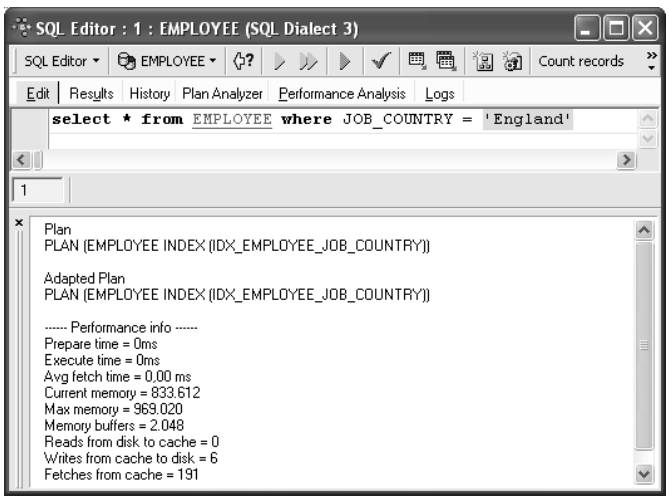


Figura 6. Consulta com o novo plano elaborado pelo otimizador

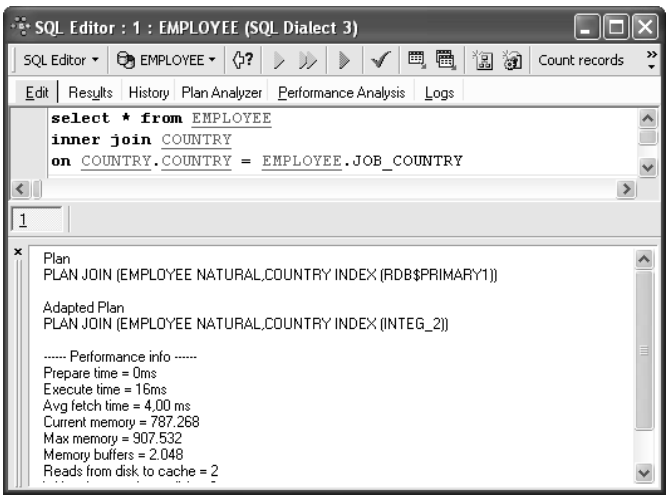


Figura 7. Consulta com Inner Join

Agora, execute a seguinte consulta:

```
select * from EMPLOYEE
inner join COUNTRY
on COUNTRY.COUNTRY = EMPLOYEE.JOB_COUNTRY
```

É possível observar que ao realizar o *Join* entre as tabelas *Employee* e *Country*, sabendo que a coluna *Country* da tabela *Country* é indexada, pois é a chave primária da tabela, e a coluna *Job_Country* da tabela *Employee* não mais.

O otimizador optou por fazer uma leitura seqüencial na tabela *Employee* e procurar os valores na coluna indexada da tabela *Country* (**Figura 7**).

Ordenação e agrupamento

Ao utilizar as cláusulas *Order By* ou *Group By* em colunas que possuem índices, o índice será utilizado para efetuar a ordenação ou agrupamento uma vez que o índice já está ordenado, não sendo preciso então realizar uma operação de *Sort*.

No exemplo da **Figura 8** é possível observar uma consulta com uma cláusula *Order By* em uma coluna que não está indexada, sendo assim, será necessário executar uma operação de *Sort*, que aumentará o tempo de execução da consulta.

Execute o seguinte código para a criação de um índice:

```
Create Index IDX_EMPLOYEE_FIRST_NAME
on EMPLOYEE (FIRST_NAME)
```

Executando novamente a consulta anterior, dessa vez com a coluna indexada, é possível observar que o otimizador fez uso do índice para realizar a ordenação, uma vez que os índices mantêm a ordenação, melhorando o tempo de execução da consulta, poupando o otimizador de realizar ordenações que consumiriam um tempo maior (**Figura 9**).

Uma vez que o otimizador do Firebird tende a utilizar índices, é importante que esses estejam bem balanceados e que a sua seletividade esteja boa. Por isso, é importante que sejam realizadas manutenções nas estruturas de índices após alterações em grandes volumes de dados, principalmente quando essas alterações são do tipo *Insert* e *Delete*.

O balanceamento dos índices é feito através da sua reconstrução utilizando seguintes comandos:

```
Alter Index nomedoindice Inactive
Alter Index nomedoindice Active
```

O primeiro desativa o índice e o segundo ativa. Ao executar esse procedimento, de desativar e ativar o índice, ele será reconstruído e, por sua vez, rebalanceado.

Quanto à seletividade dos índices, ela é importante para auxiliar o otimizador na escolha do melhor caminho durante a execução de uma consulta, uma vez que os dados sobre a seletividade dos índices fazem parte dos dados estatísticos que o otimizador utiliza para formular os planos de execução.

Para atualizar a seletividade dos índices é necessário executar o seguinte comando:

```
Set Statistics Index nomedoindice
```

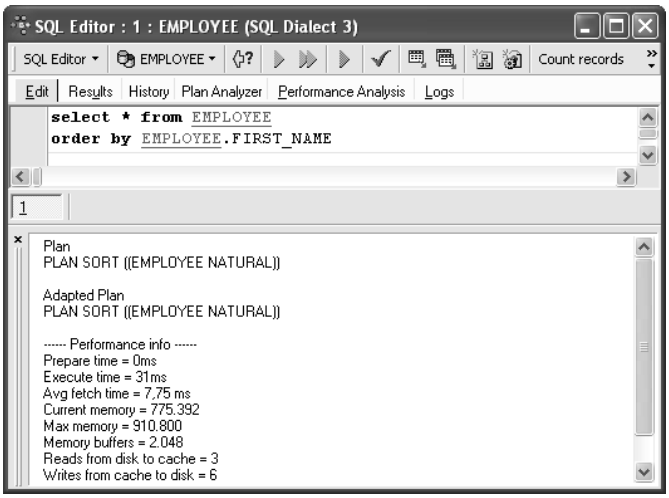


Figura 8. Cláusula de Order By em coluna não indexada

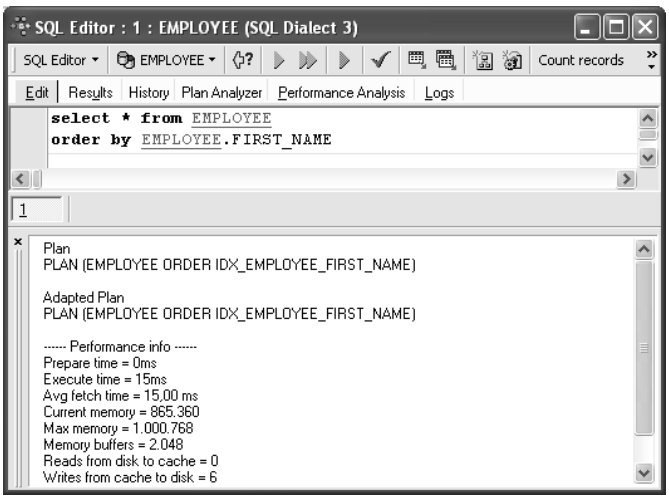


Figura 9. Consulta com cláusula de Order By na coluna indexada

Analisando os fatores apresentados anteriormente, é possível elaborar consultas que utilizam melhor os recursos que o otimizador do Firebird é capaz de fornecer.

Conclusão

Apesar de que nos dias atuais, espaço para armazenamento ser quase que ilimitado, os recursos de rede e de processamento não são. Portanto, construir aplicações otimizadas poderá contribuir muito para um bom funcionamento e utilização do sistema desenvolvido, principalmente em aplicações cliente/servidor, em que requisições de dados ao servidor são feitas constantemente.

O importante ao realizar otimizações é saber que não existe uma só maneira que trará os melhores resultados. Técnicas de otimização possuem prós e contras que devem ser comparadas, principalmente a respeito das condições do ambiente onde o sistema está inserido, como rede, plataforma cliente/servidor e capacidade de armazenamento e processamento.

Assim, para todo e qualquer recurso, é importante avaliar exaustivamente as possibilidades para que não ocorram surpresas desagradáveis ao serem colocados sistemas em produção. ■

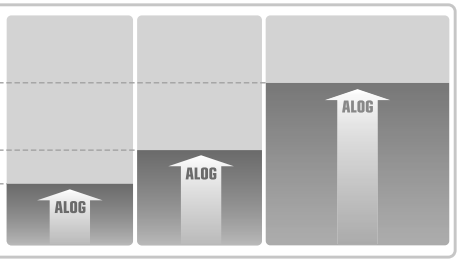
FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedica-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O **HOSTING GERENCIADO**

CAMADAS DE NEGÓCIO	DESCRIÇÃO
Processo de Negócio	Atividades cotidianas do cliente
Aplicação do Cliente	Softwares de Gestão (Ex: ERP, CRM, outros)
Ambiente da Aplicação	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange
Sistema Operacional	Windows, Linux, Unix, Solaris
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches
Rede	IP, Wan, Rede Local
Infra-estrutura Básica	Energia, HVAC, Segurança - 2N+1



ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$**1.050,00** mensais
INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

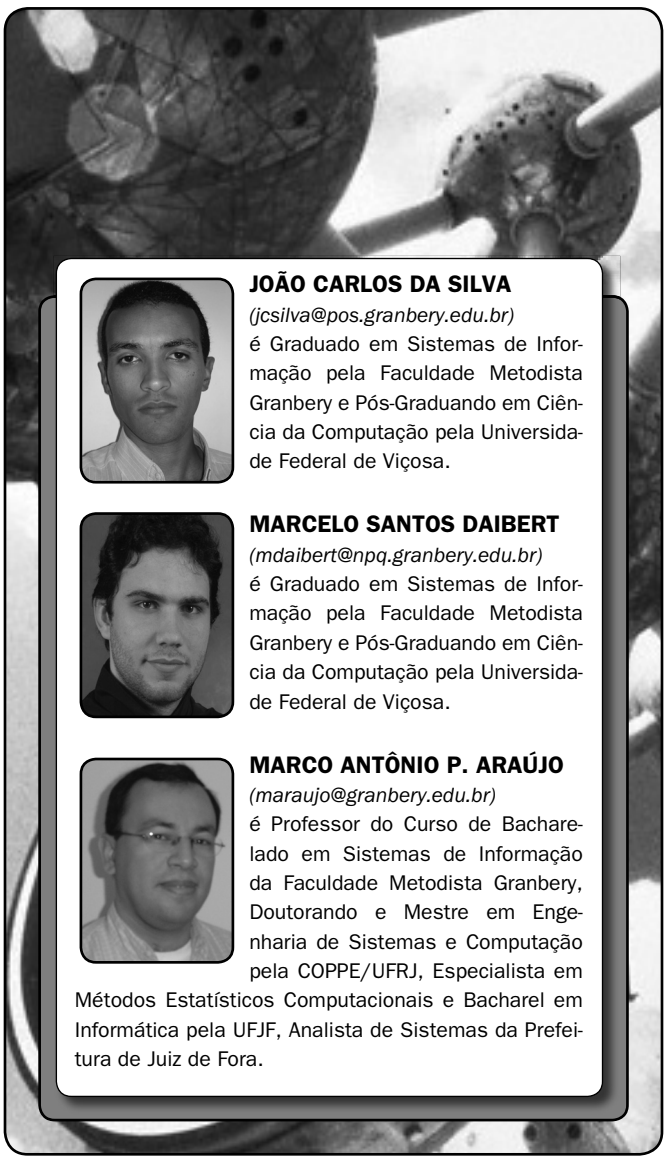
R$**2.850,00** mensais
INSTALAÇÃO IMEDIATA***

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Hosting Gerenciado | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Uma abordagem prática no desenvolvimento de aplicações OO no Delphi – Parte 2



JOÃO CARLOS DA SILVA
(jcsilva@pos.granbery.edu.br)
é Graduado em Sistemas de Informação pela Faculdade Metodista Granbery e Pós-Graduando em Ciência da Computação pela Universidade Federal de Viçosa.

MARCELO SANTOS DAIBERT
(mdaibert@npq.granbery.edu.br)
é Graduado em Sistemas de Informação pela Faculdade Metodista Granbery e Pós-Graduando em Ciência da Computação pela Universidade Federal de Viçosa.

MARCO ANTÔNIO P. ARAÚJO
(maraujo@granbery.edu.br)
é Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Como discutido na edição anterior, na primeira parte deste artigo, o paradigma orientado a objetos de desenvolvimento de software está cada vez mais se tornando o padrão de desenvolvimento. No entanto, os reais benefícios da orientação a objetos são efetivamente alcançados quando se possui domínio efetivo de conceitos da orientação a objetos e da linguagem de programação utilizada, além de assuntos correlacionados, como padrões de projeto e persistência.

Desta forma, na primeira parte deste artigo foi apresentado um estudo de caso utilizado para exemplificar o desenvolvimento de software orientado a objetos, além da definição de todas as classes, métodos e atributos do modelo. Foi apresentado ainda o conceito de mapeamento objeto-relacional e como utilizar esta abordagem no escopo do estudo de caso apresentado. Por fim foi definida a estratégia de conexão com o banco de dados, além de ter sido apresentada a definição das classes responsáveis pela manipulação e persistência dos objetos (Classes DAO).

Nesta segunda parte é apresentada a implementação da estratégia de persistência de objetos utilizada, além de exemplos e codificação de persistência e manipulação dos objetos no banco de dados relacional.

Classes de Persistência

Ao final da primeira parte deste artigo foi definida a estratégia de persistência e manipulação dos objetos. Foi possível apresentar o padrão de projeto DAO (*Data Access Object*), que busca estreitar os benefícios da orientação a objetos com a maturidade do banco de dados relacional.

A seguir serão tratadas as implementações dos métodos de persistência das classes *FuncionarioDiaristaDAO* e *FuncionarioDAO*, cobrindo assim um exemplo completo envolvendo um funcionário diarista, tratando inclusive a associação desse com um departamento, conforme definido no modelo de domínio.

Para a definição dos demais métodos das classes DAO, cujos códigos são semelhantes aos demonstrados, basta realizar poucas modificações, principalmente nas expressões em SQL codificadas.

Na **Listagem 1** está definido o *Persistir* da classe *FuncionarioDiaristaDAO*. Esse método é responsável por persistir as informações obtidas de um objeto da classe de domínio *FuncionarioDiarista* e armazená-las no banco de dados.

Nessa listagem, observa-se na linha 7 a criação de uma instância da classe *TdtmdlDados*, que é o Data Module responsável pela comunicação com o banco de dados. O *Persistir* no exemplo, realiza a dupla função de incluir um novo objeto no banco de dados, como também atualizar um objeto já existente.

Para isso, o critério de decisão para gerar as expressões SQL correspondentes a cada caso é a existência ou não de um OID do objeto. Caso o objeto ainda não possua um OID, conforme testado na linha 9, o bloco de código das linhas 10 a 17 é executado, gerando um novo OID, atribuindo e executando uma expressão SQL que garanta que somente a estrutura da tabela seja retornada e, por fim, o *ClientDataSet* é colocado em modo de inserção, conforme a linha 15.

Caso o objeto possua um OID, significa que a operação de persistência é uma atualização. Nesse caso, os dados do objeto são retornados através do *ClientDataSet* e o mesmo é colocado em modo de edição. A partir da linha 25 os dados são atribuídos ao *ClientDataSet* usando os métodos *get* do objeto passado por parâmetro, e o *ClientDataSet* é atualizado invocando o *Post*.

Caso a modificação ocorra com sucesso, o resultado de execução do *Persistir* é verdadeiro, encerrando sua execução. Ao final, o *ClientDataSet* é fechado e o seu objeto é destruído. O *Obter* da classe *FuncionarioDiaristaDAO* é definido na **Listagem 2**. Trata-se de um método de classe que recebe como parâmetro o OID de um funcionário diarista para recuperação do mesmo.

Esse método, instancia um novo objeto da classe *TdtmdlDados* (Data Module) e atribui a expressão SQL correspondente à recuperação do objeto desejado. Caso algum registro seja retornado, conforme teste realizado na linha 11, os valores constantes no *ClientDataSet* são atribuídos a um novo objeto da classe *FuncionarioDiarista*.

Nota-se que um novo objeto da classe *Departamento* também é criado e, na linha 17, os dados do departamento associado ao funcionário são recuperados, atribuídos ao objeto *Departamento* associado ao funcionário.

Um cuidado especial deve ser tomado nesse ponto, pois caso haja navegabilidade bidirecional definida no modelo de classes do domínio, ou seja, de um funcionário se obtém o seu departamento bem como de um departamento se obtém os seus funcionários, pode-se colocar o sistema em *loop* infinito. Na situação anterior, caso o *Obter* da classe *Departamento* recuperasse objetos de funcionários automaticamente, causaria o *loop* pois, ao recuperar o primeiro funcionário, esse recuperaria o seu departamento associado que por sua vez recuperaria novamente funcionários, e assim sucessivamente.

Listagem 1. Definição do Persistir da classe FuncionarioDiaristaDAO

```
1. function FuncionarioDiaristaDAO.Persistir(
2.    pFuncDia: FuncionarioDiarista): Boolean;
3. var
4.    vDataModule: TdtmdlDados;
5. begin
6.    result := False;
7.    vDataModule := TdtmdlDados.Create(nil);
8.    vDataModule.clntdtstPrincipal.Close;

9.    if pFuncDia.getOID = '' then
10.   begin
11.     pFuncDia.setOID(self.GerarOID);
12.     vDataModule.sqldtstPrincipal.CommandText :=
13.       'select OID, Nome, Tipo, Departamento, NumDias,'+
           ' ValorDia from FUNCIONARIO where OID is null';
14.     vDataModule.clntdtstPrincipal.Open;
15.     vDataModule.clntdtstPrincipal.Insert;
16.     vDataModule.clntdtstPrincipal.FieldByName(
          'OID').AsString := pFuncDia.getOID;
17.   end

18.   else
19.   begin
20.     vDataModule.sqldtstPrincipal.CommandText :=
21.       ' select OID, Nome, Tipo, Departamento, '+
          'NumDias, ValorDia from FUNCIONARIO where OID = ''' +
            pFuncDia.getOID + '''';
22.     vDataModule.clntdtstPrincipal.Open;
23.     vDataModule.clntdtstPrincipal.Edit;
24.   end;

25.   vDataModule.clntdtstPrincipal.FieldByName(
        'NOME').AsString := pFuncDia.getNome;
26.   vDataModule.clntdtstPrincipal.FieldByName(
        'TIPO').AsString := 'D';
27.   vDataModule.clntdtstPrincipal.FieldByName(
        'DEPARTAMENTO').AsString := pFuncDia.getDepartamento.getOID;
28.   vDataModule.clntdtstPrincipal.FieldByName(
        'NUMDIAS').AsInteger := pFuncDia.getNumDias;
29.   vDataModule.clntdtstPrincipal.FieldByName(
        'VALORDIA').AsCurrency := pFuncDia.getValorDia;
30.   vDataModule.clntdtstPrincipal.Post;

31.   if vDataModule.clntdtstPrincipal.ApplyUpdates(-1) = 0 then
32.   begin
33.     result := True;
34.   end;
35.   vDataModule.clntdtstPrincipal.Close;
36.   vDataModule.Free;
37.end;
```

Listagem 2. Definição do Obter da classe FuncionarioDiaristaDAO

```
1.class function FuncionarioDiaristaDAO.Obter(pOID: string):
    FuncionarioDiarista;
2.var
3.  vDataModule: TdtmdlDados;
4.  vDepartamento: DepartamentoDAO;
5.begin
6.  vDataModule := TdtmdlDados.Create(nil);
7.  vDataModule.clntdtstPrincipal.Close;
8.  vDataModule.sqldtstPrincipal.CommandText :=
9.  ' select OID, Descricao from FUNCIONARIO where OID = ''' + pOID +
       ''''' + ' AND Tipo=''D''';
10.  vDataModule.clntdtstPrincipal.Open;

11.  if vDataModule.clntdtstPrincipal.RecordCount > 0 then
12.  begin
13.    result := FuncionarioDiarista.Create;
14.    vDepartamento := DepartamentoDAO.Create;
15.    result.setOID(vDataModule.clntdtstPrincipal.FieldByName(
         'OID').AsString);
16.    result.setNome(vDataModule.clntdtstPrincipal.FieldByName(
         'NOME').AsString);
17.    result.setDepartamento(vDepartamento.Obter(
         vDataModule.clntdtstPrincipal.FieldByName('DEPARTAMENTO').AsString));
18.    result.setNumDias(vDataModule.clntdtstPrincipal.FieldByName(
         'NUMDIAS').AsInteger);
19.    result.setValorDia(vDataModule.clntdtstPrincipal.FieldByName(
         'VALORDIA').AsCurrency);
20.    vDepartamento.Free;
21.  end

22.  else
23.  begin
24.    result := nil;
25.  end;
26.  vDataModule.clntdtstPrincipal.Close;
27.  vDataModule.Free;
28.end;
```

Para evitar essa armadilha no *Obter*, pode-se adicionar um parâmetro que determine a recuperação automática ou não de objetos associados. Podem ser avaliadas outras formas mais sofisticadas de evitar essa recursividade. No caso de navegação bidirecional, podem ser utilizadas, por exemplo, de técnicas de *lazy loading* (carregamento tardio).

Já na **Listagem 3**, é apresentado o código referente ao método *ObterFuncionariosPorOIDDepto* da classe *FuncionarioDAO*, que é responsável por recuperar todos os objetos de um determinado *Departamento*. Para isso, utiliza-se uma lista (*TList*) para armazenar os objetos.

Cada objeto é instanciado de acordo com o tipo de funcionário obtido do banco de dados, ou seja, funcionário diarista ou mensalista. Ainda os departamentos associados são recuperados e atribuídos aos respectivos objetos de funcionários.

É definido na **Listagem 4** o *ObterTodos* da classe *FuncionarioDAO*, que é similar ao método analisado na **Listagem 3**, sendo que nesse caso não é considerado um departamento específico. Sendo assim, todos os objetos de funcionários são recuperados.

Listagem 3. ObterFuncionariosPorOIDDepto da classe FuncionarioDAO

```
class function FuncionarioDAO.
  ObterFuncionariosPorOIDDepto(
  pOIDDepartamento: string): TList;
var
  vDataModule: TdtmdlDados;
  vLista: TList;
  vFuncionarioDiarista: FuncionarioDiarista;
  vFuncionarioMensalista: FuncionarioMensalista;
  vDepartamento: DepartamentoDAO;
begin
  vDataModule := TdtmdlDados.Create(nil);
  vDataModule.clntdtstPrincipal.Close;
  vDataModule.sqldtstPrincipal.CommandText :=
    ' select OID, Nome, Departamento, Tipo, '+
    'ValorMes, NumDias, ValorDia from FUNCIONARIO '+
    'WHERE Departamento = ''' +
    pOIDDepartamento + '''';
  vDataModule.clntdtstPrincipal.Open;
  if vDataModule.clntdtstPrincipal.RecordCount>0 then
  begin
    vLista := TList.Create;
    vDataModule.clntdtstPrincipal.First;
    while not vDataModule.clntdtstPrincipal.EOF do
    begin
      if vDataModule.clntdtstPrincipal.FieldByName(
        'Tipo').AsString = 'D' then
      begin
        vFuncionarioDiarista := FuncionarioDiarista.Create;
        vDepartamento := DepartamentoDAO.Create;
        vFuncionarioDiarista.setOID(
          vDataModule.clntdtstPrincipal.FieldByName(
          'OID').AsString);
        vFuncionarioDiarista.setNome(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Nome').AsString);
        vFuncionarioDiarista.setDepartamento(
          vDepartamento.Obter(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Departamento').AsString));
        vFuncionarioDiarista.setNumDias(
          vDataModule.clntdtstPrincipal.FieldByName(
          'NumDias').AsInteger);
        vFuncionarioDiarista.setValorDia(
          vDataModule.clntdtstPrincipal.FieldByName(
          'ValorDia').AsCurrency);
        vLista.Add(vFuncionarioDiarista);
        vDepartamento.Free;
      end
      else if
        vDataModule.clntdtstPrincipal.FieldByName(
          'Tipo').AsString = 'M' then
      begin
        vFuncionarioMensalista := FuncionarioMensalista.Create;
        vDepartamento := DepartamentoDAO.Create;
        vFuncionarioMensalista.setOID(
          vDataModule.clntdtstPrincipal.FieldByName(
          'OID').AsString);
        vFuncionarioMensalista.setNome(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Nome').AsString);
        vFuncionarioMensalista.setDepartamento(
          vDepartamento.Obter(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Departamento').AsString));
        vFuncionarioMensalista.setValorMes(
          vDataModule.clntdtstPrincipal.FieldByName(
          'ValorMes').AsCurrency);
        vLista.Add(vFuncionarioMensalista);
        vDepartamento.Free;
      end;
      vDataModule.clntdtstPrincipal.Next;
    end;
  end;
  result := vLista;
  vDataModule.clntdtstPrincipal.Close;
  vDataModule.Free;
end;
```

Listagem 4. ObterTodos da Classe FuncionarioDAO

```
class function FuncionarioDAO.ObterTodos: TList;
var
  vDataModule: TdtmdlDados;
  vLista: TList;
  vFuncionarioDiarista: FuncionarioDiarista;
  vFuncionarioMensalista: FuncionarioMensalista;
  vDepartamento: DepartamentoDAO;
begin
  vDataModule := TdtmdlDados.Create(nil);
  vDataModule.clntdtstPrincipal.Close;
  vDataModule.sqldtstPrincipal.CommandText :=
    ' select OID, Nome, Departamento, Tipo, '+
    'ValorMes, NumDias, ValorDia from FUNCIONARIO';
  vDataModule.clntdtstPrincipal.Open;
  if vDataModule.clntdtstPrincipal.RecordCount>0 then
  begin
    vLista := TList.Create;
    vDataModule.clntdtstPrincipal.First;
    while not vDataModule.clntdtstPrincipal.EOF do
    begin
      if vDataModule.clntdtstPrincipal.FieldByName(
        'Tipo').AsString = 'D' then
      begin
        vFuncionarioDiarista :=
          FuncionarioDiarista.Create;
        vDepartamento := DepartamentoDAO.Create;
        vFuncionarioDiarista.setOID(
          vDataModule.clntdtstPrincipal.FieldByName(
          'OID').AsString);
        vFuncionarioDiarista.setNome(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Nome').AsString);
        vFuncionarioDiarista.setDepartamento(
          vDepartamento.Obter(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Departamento').AsString));
        vFuncionarioDiarista.setNumDias(
          vDataModule.clntdtstPrincipal.FieldByName(
          'NumDias').AsInteger);
        vFuncionarioDiarista.setValorDia(
          vDataModule.clntdtstPrincipal.FieldByName(
          'ValorDia').AsCurrency);
        vLista.Add(vFuncionarioDiarista);
        vDepartamento.Free;
      end
      else if
        vDataModule.clntdtstPrincipal.FieldByName(
          'Tipo').AsString = 'M' then
      begin
        vFuncionarioMensalista :=
          FuncionarioMensalista.Create;
        vDepartamento := DepartamentoDAO.Create;
        vFuncionarioMensalista.setOID(
          vDataModule.clntdtstPrincipal.FieldByName(
          'OID').AsString);
        vFuncionarioMensalista.setNome(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Nome').AsString);
        vFuncionarioMensalista.setDepartamento(
          vDepartamento.Obter(
          vDataModule.clntdtstPrincipal.FieldByName(
          'Departamento').AsString));
        vFuncionarioMensalista.setValorMes(
          vDataModule.clntdtstPrincipal.FieldByName(
          'ValorMes').AsCurrency);
        vLista.Add(vFuncionarioMensalista);
        vDepartamento.Free;
      end;
      vDataModule.clntdtstPrincipal.Next;
    end;
  end;
  result := vLista;
  vDataModule.clntdtstPrincipal.Close;
  vDataModule.Free;
end;
```

Por fim, na **Listagem 5**, é apresentado o *Destruir* da classe *FuncionarioDAO*. Esse método recupera um objeto específico da classe *Funcionario*, não determinando se o mesmo é diarista ou mensalista e faz sua exclusão.

Uma instrução SQL de seleção é executada para recuperar as informações do objeto e, em seguida, o registro do *ClientDataSet* é excluído, sendo então submetida a modificação ao banco de dados.

Listagem 5. Destruir da Classe FuncionarioDAO

```
function FuncionarioDAO.Destruir(
  pOID: string): Boolean;
var
  vDataModule: TdtmdlDados;
begin
  result := False;
  if pOID<>'' then
  begin
    vDataModule := TdtmdlDados.Create(nil);
    vDataModule.clntdtstPrincipal.Close;
    vDataModule.sqldtstPrincipal.CommandText :=
      ' select OID from FUNCIONARIO where OID = ''' + pOID + ''' ;
    vDataModule.clntdtstPrincipal.Open;
    vDataModule.clntdtstPrincipal.Delete;
    if vDataModule.clntdtstPrincipal.ApplyUpdates(-1) = 0 then
    begin
      result := True;
    end;
    vDataModule.clntdtstPrincipal.Close;
    vDataModule.Free;
  end;
end;
```

Existem diversas formas de realizar as operações de persistência utilizando essa abordagem. É necessário, porém, destacar que a separação dessas operações em uma camada pode ser feita de maneira simples como demonstrado aqui.

Utilizando classes DAO, pode-se encapsular as expressões SQL, isolando essas do restante da aplicação, permitindo melhor controle e manutenibilidade da aplicação. Eventuais validações de dados podem ser feitas diretamente nas classes de domínio, deixando as classes DAO livres de regras de negócio, o que facilitaria futuramente uma substituição por camadas de persistência mais robustas se necessário.

Ainda, pode-se agregar o padrão de projeto MVC (*Model-View-Controller*) para separar as classes de domínio (*Model*) das questões ligadas à interface com o usuário (*View*), por meio de classes intermediárias de controle (*Controller*). Isso permite uma maior independência da camada de domínio.

Neste caso, as validações poderiam ficar a cargo da camada de controle. Em uma terceira análise, estendendo ainda mais a divisão em camadas, pode-se estabelecer uma camada própria para lógicas de negócios, ou seja, as regras específicas da aplicação estariam alocadas nessa camada, sendo o local apropriado para efetuar validações e determinar formas de funcionamento do sistema.

Persistência e Manipulação de Objetos

Para exemplificar o uso das abordagens aplicadas neste artigo, é construída uma das interfaces da aplicação, como a janela que corresponde ao cadastro de novos funcionários (**Figura 1**).

Nota: O código-fonte completo está para download.

Listagem 6. Código da interface de cadastro de funcionário

```
1.var
2.  frmFuncionario: TfrmFuncionario;
3.  vDepartamentos: TList;
4.
5.implementation
6.uses untDominio;
7.
8.procedure TfrmFuncionario.FormActivate(Sender: TObject);
9.var
10.  i: integer;
11.begin
12.  vDepartamentos:= TList.Create;
13.  vDepartamentos := Departamento.ObterTodos;
14.  for i:=0 to vDepartamentos.count-1 do
15.  begin
16.  cmbxDepartamento.Items.Add(Departamento(
       vDepartamentos[i]).getDescricao);
17.  end;
18.end;
19.procedure TfrmFuncionario.btnOKClick(Sender: TObject);
20.var
21.  func: Funcionario;
22.  depto: Departamento;
23.begin
24.  case rdgrpTipoFuncionario.ItemIndex of
25.    0: begin
26.         func := FuncionarioMensalista.Create;
27.           FuncionarioMensalista(func).SetValorMes(StrToFloat(
              edtMensalistaValorMes.Text));
28.       end;
29.    1: begin
30.         func := FuncionarioDiarista.Create;
31.         FuncionarioDiarista(func).SetValorDia(
              StrToFloat(edtDiaristaValorDia.Text));
32.         FuncionarioDiarista(func).SetNumDias(
              StrToInt(edtDiaristaNumDias.Text));
33.       end;
34.    else
35.    begin
36.      ShowMessage('Tipo de funcionário não selecionado');
37.    exit;
38.    end;
39.  end;
40.  func.SetNome(edtNome.Text);
41.  depto := Departamento.obter((Departamento(vDepartamentos[
       cmbxDepartamento.ItemIndex]).getOID));
42.  if Assigned(depto) then
43.  begin
44.    func.SetDepartamento(depto);
45.  end;
46.  func.Persistir;
47.  edtNome.Clear;
48.  cmbxDepartamento.ItemIndex := -1;
49.  rdgrpTipoFuncionario.ItemIndex := -1;
50.  edtMensalistaValorMes.Clear;
51.  edtDiaristaValorDia.Clear;
52.  edtDiaristaNumDias.Clear;
53.  edtNome.SetFocus;
54.  func.Free;
55.  depto.Free;
56.end;
57.procedure TfrmFuncionario.FormClose(
     Sender: TObject; var Action: TCloseAction);
58.begin
59.  vDepartamentos.Free;
60.end;
```

Essa interface permite informar dados de um novo funcionário bem como associá-lo a um departamento, previamente cadastrado. Ainda, pode-se definir o tipo de funcionário, sendo esse mensalista ou diarista.

Na **Listagem 6**, é apresentado o código correspondente à interface de Cadastro de Funcionários.

Na seção *var* (declaração de variáveis da unit) está declarada uma variável que armazena os departamentos que são apresentados no *ComboBox*. Para esse caso, é definida uma variável do tipo *TList* que armazena os objetos do tipo *Departamento*.

Entre as linhas 8 e 18 está o código relativo ao evento *OnActivate* do formulário. Esse evento trata do carregamento do *ComboBox* com as descrições dos departamentos. Dessa forma, na linha 13 é chamado o *ObterTodos* da classe *Departa-*

mento, que por sua vez chama o método correspondente na classe DAO. Obtida a lista de departamentos, a mesma é percorrida obtendo individualmente a descrição de cada objeto e atribuindo o valor no *ComboBox*.

A partir da linha 19 está o código que é executado ao clicar no botão OK, responsável pela confirmação da operação de persistência dos objetos. São manipulados dois objetos, o funcionário e o seu departamento associado, para isso duas variáveis são declaradas, como pode ser visto nas linhas 21 e 22.

Conforme o tipo do funcionário selecionado em tela, um objeto de uma das classes, *FuncionarioMensalista* ou *FuncionarioDiarista*, é instanciado. Posteriormente os valores informados em tela são atribuídos ao objeto da classe *Funcionario* por meio dos métodos *set*, utilizando a técnica de *typecasting* para a conversão para a subclasse correspondente.

Para a associação com *Departamento*, o objeto correspondente à opção selecionada no *ComboBox* é recuperado. Caso tenha sido recuperado o objeto, o mesmo é atribuído ao funcionário. Com isso, é chamada a função *Persistir* do *Funcionario*, acionando o método da classe DAO correspondente, via polimorfismo.

Por fim, as linhas 54 e 55 apresentam a destruição dos objetos manipulados. Ao finalizar a execução, o objeto é persistido com associação a um departamento. O último procedimento, apresentado entre as linhas 58 e 60 referencia o evento *FormClose*. Esse evento inicia quando o formulário é fechado e é responsável por destruir o objeto *vDepartamentos*.

Outro exemplo apresentado é a *interface* de visualização dos funcionários por departamento, como pode ser visto na **Figura 2**.

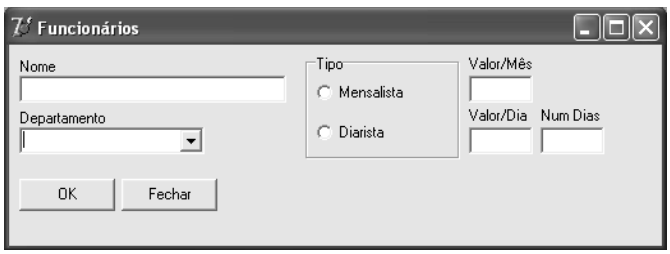


Figura 1. Interface para cadastro de funcionário

Figura 2. Interface para visualização de funcionários por departamento

Essa *interface* apresenta um *ComboBox* onde são apresentados os departamentos e um *ListView* para visualização dos funcionários do departamento selecionado. A **Listagem 7** exibe o código fonte referente a esta *interface*.

Na seção de declaração de variáveis é possível observar uma variável do tipo *TStringList* responsável por armazenar a lista de *OIDs* dos departamentos carregados no *ComboBox*. Entre as linhas 8 e 11, é apresentado o código do referente ao evento *FormCreate*. Esse evento se inicia assim que o formulário é instanciado e tem a função, neste exemplo, de instanciar o objeto definido anteriormente.

Listagem 7. Código da para visualização de funcionários por departamento

```
1.var
2.  frmFuncionariosPorDepartamento: TfrmFuncionariosPorDepartamento;
3.  vOIDDepartamentos: TStringlist;
4.
5.implementation
6.  uses untDominio;
7.
8.procedure TfrmFuncionariosPorDepartamento.FormCreate(Sender: TObject);
9.begin
10.  vOIDDepartamentos := TStringList.Create;
11.end;
12.procedure TfrmFuncionariosPorDepartamento.(Sender: TObject);
13.var
14.  vDepartamentos: TList;
15.  i: integer;
16.begin
17.  vDepartamentos:=  TList.Create;
18.  vDepartamentos := Departamento.ObterTodos;
19.  if vDepartamentos <> nil then
20.  begin
21.    for i:=0 to vDepartamentos.Count-1 do
22.    begin
23.      cmbxDepartamento.Items.Add(Departamento(
           vDepartamentos[i]).getDescricao);
24.      vOIDDepartamentos.Add(Departamento(vDepartamentos[i]).getOID);
25.    end;
26.  end;
27.  vDepartamentos.Free;
28.end;
29.procedure TfrmFuncionariosPorDepartamento.
     cmbxDepartamentoChange(Sender: TObject);
30.var
31.  vFuncionarios: TList;
32.  i: integer;
33.  vDado : String;
34.  vItem :TListItem;
35.  vTipo : String;
36.begin
37.  lstvwFuncionarios.Clear;
38.  vFuncionarios:= TList.Create;
39.  vFuncionarios := Funcionario.ObterFuncionariosPorOIDDepartamento(
       vOIDDepartamentos.Strings[cmbxDepartamento.ItemIndex]);
40.  if Assigned(vFuncionarios) then
41.  begin
42.    for i:=0 to vFuncionarios.Count-1 do
43.    begin
44.      vItem := lstvwFuncionarios.Items.Add;
45.      vItem.Caption := Funcionario(vFuncionarios[i]).getOID;
46.      if (Funcionario(vFuncionarios[i]) is
           FuncionarioDiarista) then
47.      begin
48.        vTipo := 'Diarista';
49.      end
50.      else if (Funcionario(vFuncionarios[i]) is
           FuncionarioMensalista) then
51.      begin
52.        vTipo := 'Mensalista';
53.      end;
54.      vItem.SubItems.Add(Funcionario(
           vFuncionarios[i]).getNome);
55.      vItem.SubItems.Add(vTipo);
56.      vItem.SubItems.Add('R$ '+FloatToStr(
           Funcionario(vFuncionarios[i]).calcularSalario()));
57.    end;
58.  end;
59.  vFuncionarios.Free;
60.end;
61.procedure TfrmFuncionariosPorDepartamento.FormClose(Sender: TObject;
     var Action: TCloseAction);
62.begin
63.  vOIDDepartamentos.Free;
64.end;
```

Já entre as linhas 12 e 28, é apresentado o evento *FormActivate* do formulário. Esse evento é iniciado após o *FormCreate* e tem a função de carregar e popular o *ComboBox* e o objeto *TStringList* com os departamentos persistidos no banco de dados. Essa operação é semelhante à apresentada na interface de cadastro de funcionários.

O evento que trata as modificações do *ComboBox* é exibido entre as linhas 29 e 60. Esse evento é o responsável por buscar no banco de dados os funcionários de cada departamento selecionado e popular o *ListView* com seus dados.

Na linha 39 é invocado o método de classe *ObterFuncionariosPorOIDDepartamento* responsável por retornar um *TList* de funcionários de determinado departamento. Esse departamento selecionado é passado como parâmetro à função, através do seu OID obtido a partir do índice do *ComboBox*. Após, esse *TList* retornada é "varrido", populando o *ListView*. Ao final, o *TList* é destruído.

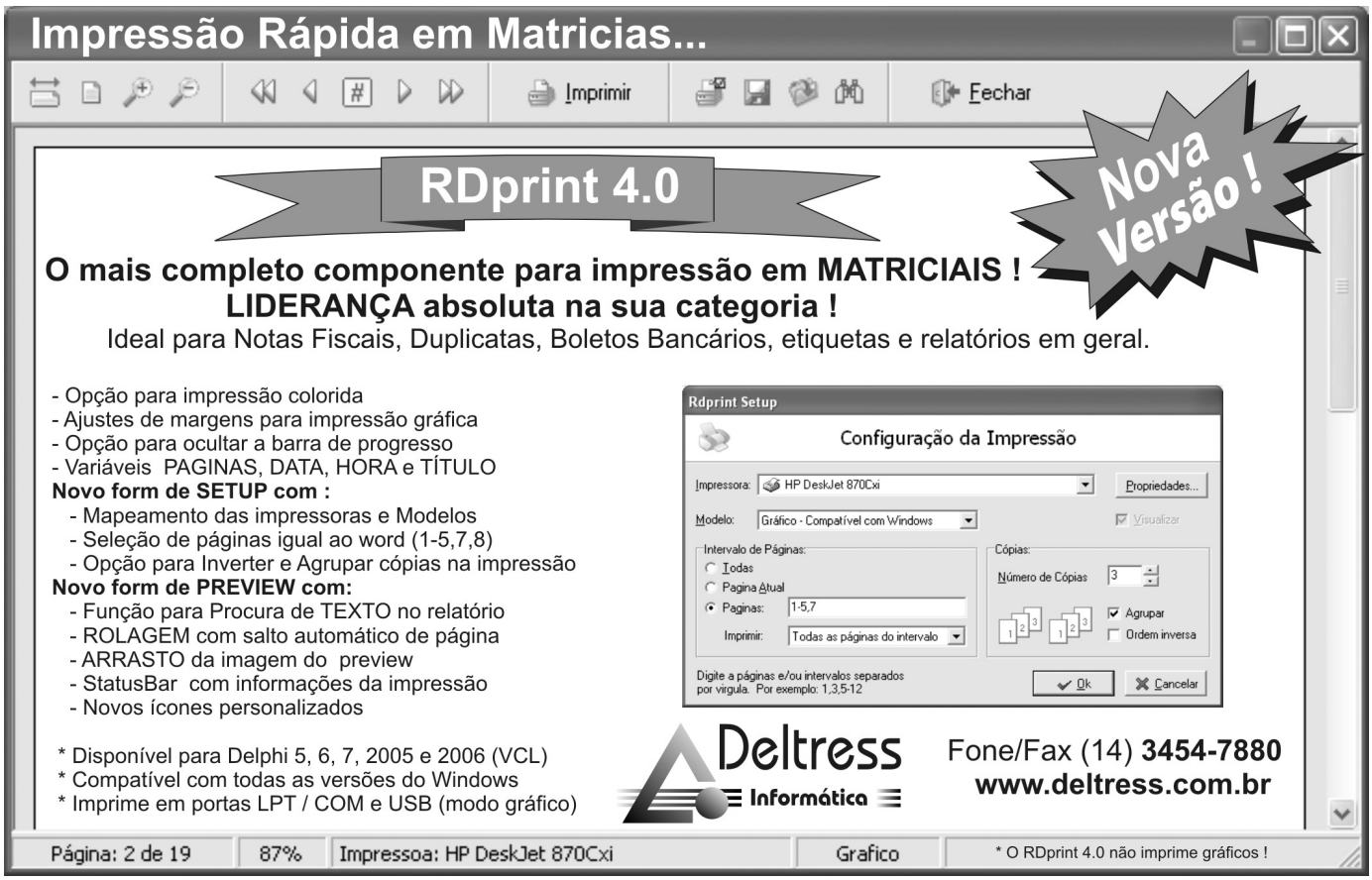
Das linhas 61 à 64 é apresentado o evento *FormClose* responsável por destruir o objeto *TStringList* definido na seção de declaração de variáveis do formulário.

Conclusão

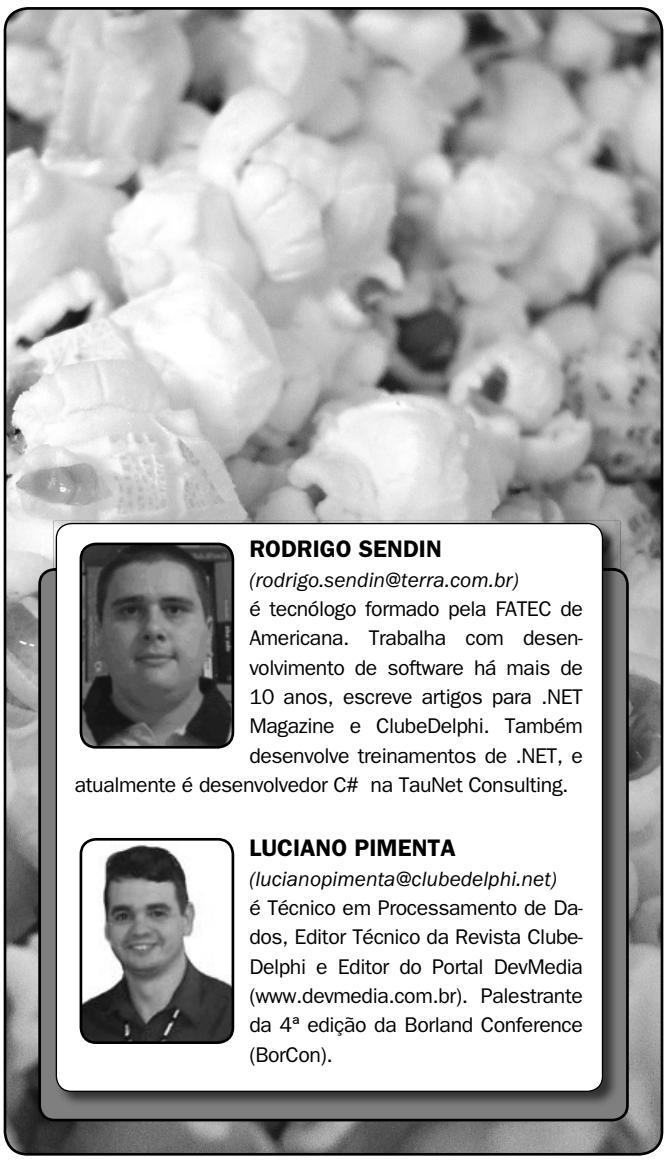
Este artigo teve como objetivo abordar de forma prática a utilização do paradigma Orientado a Objetos no contexto do desenvolvimento de software no ambiente Delphi e de técnicas aliadas, como mapeamento objeto relacional e utilização de padrões de desenvolvimento.

Foi possível observar na prática como realizar de forma simples a persistência e manipulação de objetos utilizando um banco de dados relacional, apoiado por padrões de desenvolvimento, como o padrão de projeto DAO (*Data Access Object*), que encapsula todo o contexto de conexão e comunicação com o banco de dados.

Atualmente, essa abordagem tem uma grande aceitação, uma vez que os bancos de dados Orientados a Objetos ainda não são o padrão de mercado, tornando possível a união entre os conceitos da Orientação a Objetos, com o desempenho, maturidade e confiabilidade dos bancos de dados relacionais. ■



Pesquisas estilo Lookup usando janelas popup na Web



RODRIGO SENDIN
(rodrigo.sendin@terra.com.br)
é tecnólogo formado pela FATEC de Americana. Trabalha com desenvolvimento de software há mais de 10 anos, escreve artigos para .NET Magazine e ClubeDelphi. Também desenvolve treinamentos de .NET, e atualmente é desenvolvedor C# na TauNet Consulting.

LUCIANO PIMENTA
(lucianopimenta@clubedelphi.net)
é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi e Editor do Portal DevMedia (www.devmedia.com.br). Palestrante da 4ª edição da Borland Conference (BorCon).

Todo sistema que trabalhe com um banco de dados, com tabelas relacionadas, precisa disponibilizar pesquisas para busca de registros, como por exemplo, uma pesquisa para encontrar um determinado cliente ou os produtos de um pedido.

Quando estamos desenvolvendo um sistema para a Web temos a mesma necessidade, porém muitas vezes ficamos limitados pelos controles disponíveis. Uma boa alternativa é o uso de controles como o *DropDownList* ou *ListBox*, porém algumas pesquisas precisam de uma interface mais detalhada, principalmente quando a tabela onde faremos a pesquisa tiver uma grande quantidade de registros.

Neste artigo veremos como desenvolver pesquisas no estilo *lookup*, para atender essa necessidade tão comum do nosso dia-a-dia.

Web Application

Abra o Delphi 2005/2006 e crie um novo projeto ASP.NET, chamado "Lookups". Vamos desenvolver uma página onde simularemos o cadastro de um empregado. Para isso, utilizaremos o banco *Employee* que acompanha a instalação do Firebird.

Caso ainda não tenha instalado o provider do Firebird, faça-o antes de continuar o exemplo. Na página criada por *default*, desenhe uma interface como demonstrado na **Figura 1**, onde temos um formulário simples para cadastro de empregados.

ClubeDelphi PLUS

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como trabalhar com JavaScript em aplicações ASP.NET para utilizar telas de consultas, atualizando a consulta na página chamadora.

www.devmedia.com.br/articles/viewcomp.asp?comp=4270

Observe que temos dois *Buttons* no formulário, um para o campo *Hire Date* e outro para o *Dept No*. Eles serão utilizados para chamarmos as janelas de pesquisa do nosso exemplo. Uma observação importante antes de continuar é que, como o foco do artigo é a pesquisa *lookup*, não vamos nos preocupar com validações e rotinas de inclusão.

Pesquisa de Departamentos

Uma pesquisa do tipo *lookup* nada mais é do que uma outra janela que será apresentada para a consulta da tabela em questão. Portanto nossa interface de pesquisa de clientes será uma outra página ASPX. Para isso, vamos incluir uma nova página no projeto e chamá-la de "Department.aspx". Adicione um *FbConnection* e faça a conexão com o banco *Employee.fdb* (**Figura 2**).

Arraste um *FbCommand* para a página que acabamos de criar. Adicione também um *DataGrid*. No *FbCommand*, adicione o seguinte comando SQL na propriedade *CommandText*:

```
select DEPT_NO, DEPARTMENT
from DEPARTMENT
```

Caso deseje e é recomendado, você pode parametrizar a pesquisa, realizando assim uma busca por somente parte do campo *Department*. Não esqueça de fazer a ligação do *FbCommand* ao *FbConnection*, através da propriedade *Connection*.

Abra as configurações do *DataGrid* e adicione uma coluna do tipo *Button Column>Select*. O layout do formulário deverá estar parecido com o da **Figura 3**.

No *Page_Load* da página adicione o código da **Listagem 1**, para preencher o *DataGrid*.

Volte ao *WebForm1.aspx*, dê um duplo clique no botão de *lookup* do departamento e insira o código da **Listagem 2**.

Veja que foi criado um comando *JavaScript* que abre uma nova janela do browser. Observe que através do *window.open* podemos delimitar o tamanho da janela, definir que a nova janela não possuirá barras de rolagem, não poderá ser redimensionada e nem terá opções de menu.

Listagem 1. Preenchendo o DataGrid de departamentos

```
if not IsPostBack then
begin
  FbConnection1.Open;
  try
    DataGrid1.DataSource := FbCommand1.ExecuteReader;
    DataGrid1.DataKeyField := 'DEPT_NO';
    DataGrid1.DataBind;
  finally
    FbConnection1.Close;
  end;
end;
```

Listagem 2. Evento click do botão de departamentos

```
uses System.Text;
...
var
  script: StringBuilder;
begin
  script := StringBuilder.Create;
  script.Append('<script language="JavaScript">');
  script.Append('window.open("Department.aspx","", ');
  script.Append(
    '"resizable=no, menubar=no,scrollbars=yes, '+
       'width=450,height=450")');
  script.Append('</script>');
  RegisterClientScriptBlock('client', .ToString);
end;
```

Nota: Para saber mais sobre as configurações (parâmetros) do *window.open* acesse: *msdn.microsoft.com/workshop/author/dhtml/reference/methods/open_0.asp.*

O *JavaScript* é executado através do *RegisterClientScriptBlock*, que temos no objeto *Page*. Uma observação muito importante é que essa janela é aberta como um *popup*, portanto é necessário que você habilite o uso de *popups* de cada cliente que utilizará a aplicação.

Para finalizar nosso primeiro exemplo, volte à página *Department.aspx*. Clique no *DataGrid* e acesse o evento *ItemCommand*. Adicione o código da **Listagem 3**. Veja que estamos novamente disparando um *JavaScript* para retornar o código e o nome do cliente.

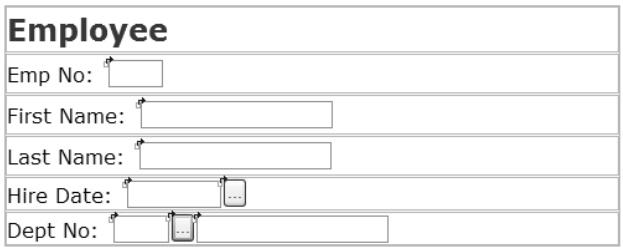


Figura 1. Formulário para cadastro de empregados

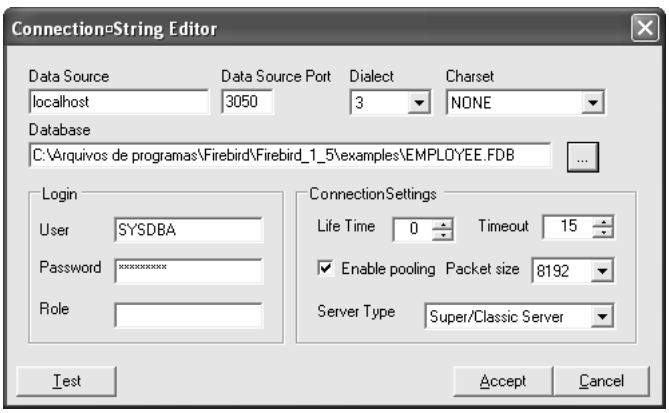


Figura 2. Conectando ao banco Employee.fdb

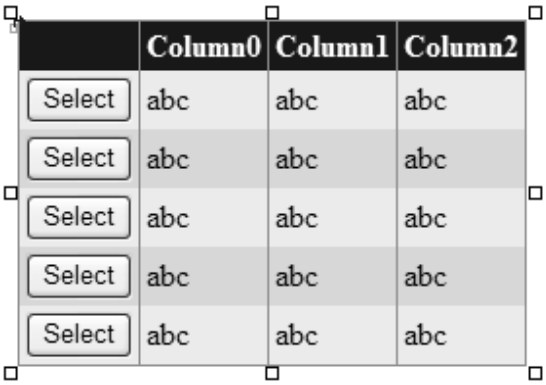


Figura 3. Formulário de pesquisa de departamentos

Testando a pesquisa Lookup de departamentos

Salve, compile e execute o projeto para que possamos testá-lo. Clique no botão do campo *Dept No* e veja que uma nova janela é aberta trazendo o *DataGrid* com todos os clientes da tabela *Department* (**Figura 4**).

Escolha um departamento qualquer e clique no respectivo botão *Select*. Veja que o código e o nome do cliente são retornados ao formulário de inclusão de pedidos.

Calendário

Uma outra pesquisa *lookup* muito comum é o calendário para campos do tipo Data. Neste exemplo, vamos utilizar o controle de calendário disponível no ASP.NET e vamos chamá-lo em uma outra janela, assim como fizemos com a pesquisa de clientes.

Adicione uma nova página, dando o nome de "Calendario.aspx". Assim como mostra a **Figura 5**, crie uma tabela, arraste um controle do tipo *Calendar* e dois *Buttons* ("btnCancel" e "btnOK"). Dê um duplo clique no *Cancel* e digite o código da **Listagem 4**.

Estamos simplesmente fechando o formulário através de um *JavaScript* (*Self.close*). Para completar, volte ao design e dê um duplo clique no botão OK. Inclua o código da **Listagem 5** e veja que similar ao que fizemos na pesquisa de departamentos, estamos preenchendo o *txtHireDate* com a data selecionada no *Calendar*.

Novamente estamos utilizando um *JavaScript* e registrando o comando através do *RegisterClientScriptBlock*. A diferença em relação ao exemplo anterior, é que formatamos a data para ser apresentada no formato dd/MM/yyyy.

Basta agora, codificarmos a chamada do calendário na página *WebForm1.aspx*. Na página, dê um duplo clique no botão de *lookup* do campo *Hire Date* e adicione o código da **Listagem 6**.

Observe que novamente, através do *window.open*, estamos abrindo uma janela do browser com tamanho delimitado. Faça um teste. Salve, compile, execute e veja como mostra a **Figura 6**, que a página do calendário aparece como um *popup*. Basta selecionar uma data, clicar em OK e a mesma é preenchida no *txtHireDate*.

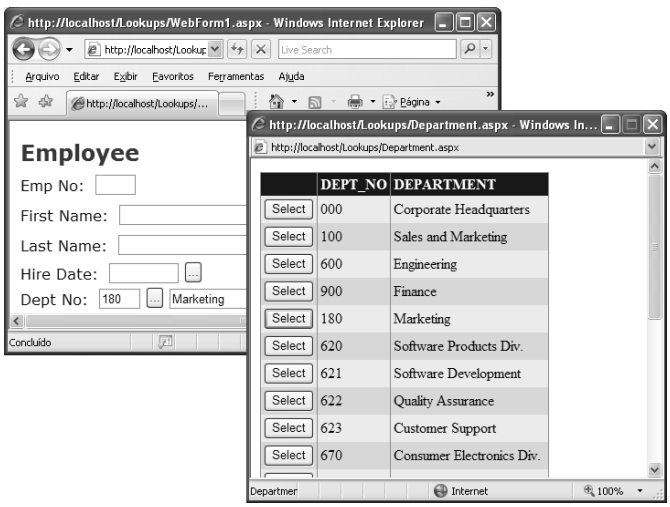


Figura 4. Testando a pesquisa Lookup de departamentos

Figura 5. Formulário Calendario.aspx

Conclusão

Como foi visto, é muito simples trabalhar com pesquisas no estilo *lookup*. Com um pouco de *JavaScript* é possível abrir as janelas

Listagem 3. Evento ItemCommand do DataGrid

```
var
  script: StringBuilder;
begin
  if e.CommandName = 'Select' then
  begin
    { atente para o nome dos controles (TextBox)
      adicionados no WebForm1.aspx }
    script := StringBuilder.Create;
    script.Append('<script language="JavaScript">');
    script.Append('window.opener.document.forms[0]');
    script.Append('["txtDeptNo"].value="' +
      DataGrid1.DataKeys[e.Item.ItemIndex].ToString +
      '";');
    script.Append('window.opener.document.forms[0]');
    script.Append('["txtDepartment"].value="' +
      e.Item.Cells[2].Text + '";');
    script.Append('window.close(); </script>');
    RegisterClientScriptBlock('client',
      script.ToString);
  end;
```

Listagem 4. Evento click do botão Cancel

```
var
  script: StringBuilder;
begin
  script := StringBuilder.Create;
  script.Append('<script language="JavaScript">');
  script.Append('window.close(); </script>');
  RegisterClientScriptBlock('client', script.ToString);
end;
```

Listagem 5. Evento click do botão OK

```
var
  script: StringBuilder;
  aData: string;
begin
  aData := System.&String.Format('{0:dd/MM/yyyy}',
    Calendar1.SelectedDate);
  script := StringBuilder.Create;
  script.Append('<script language="JavaScript">');
  script.Append('window.opener.document.forms[0]');
  script.Append('["txtHireDate"].value="' + aData + '";');
  script.Append('window.close(); </script>');
  RegisterClientScriptBlock('client', script.ToString);
end;
```

Listagem 6. Chamando a página do calendário

```
var
  script: StringBuilder;
begin
  script := StringBuilder.Create;
  script.Append('<script language="JavaScript">');
  script.Append('window.open("Calendario.aspx", "",');
  script.Append('"resizable=no, menubar=no,scrollbars=no, '+
    'width=250,height=250")');
  script.Append('</script>');
  RegisterClientScriptBlock('client', script.ToString);
end;
```

na forma de *popups*, delimitar o tamanho dessas e retirar funcionalidades como *scrollbars*, redimensionamento e opções de menu. Utilizando essa mesma técnica podemos criar telas de pesquisa mais avançadas, com o uso de filtros e múltipla seleção.

Essa é a maneira mais simples de criar pesquisas *lookups* no ASP.NET. O efeito colateral dessa solução é o uso de *popups*, que muitas vezes são bloqueados nos browser clientes. É possível criar consultas desse mesmo tipo sem o uso de *popups*, porém necessitaríamos de uma codificação mais avançada em *JavaScript* ou utilizarmos alguns controles Ajax. Um grande abraço a todos e até a próxima. ■

ClubeDelphi PLUS

Acesse agora o mesmo o Portal .net Plus (www.devmedia.com.br/dotnet) e assista a várias aulas de Fabrício Desbessel, que mostra a utilização de Ajax em aplicações ASP.NET com Delphi 2006.

www.devmedia.com.br/articles/viewcomp.asp?comp=4068

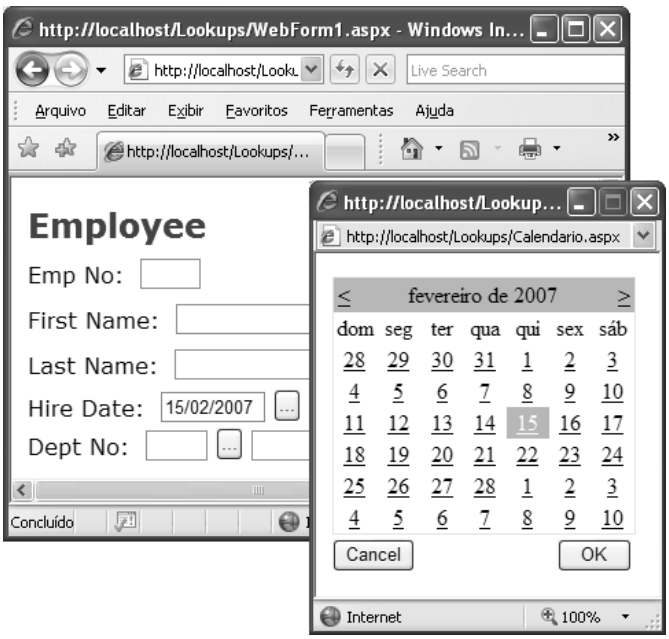


Figura 6. Testando pesquisa Lookup de calendário

Fale conosco!

Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.

Na Central de Atendimento você:

- Consulta as perguntas mais frequentes

- Renova por boleto ou cartão de crédito

- Consulta e altera seus dados cadastrais

- Consulta a data de envio de cada exemplar

Dados Cadastrais	
Login:	Ricsilva
Senha:	rsilva2006
Nome:	Ricardo da Silva
Empresa:	Minha Empresa
E-Mail:	ricsilva@email.com
E-Mail Trab:	workgroup@email.com
Endereço:	Rua das Marrecas, 32
Bairro:	Centro
Cidade:	Rio de Janeiro
Estado:	RJ
País:	Brasil
Cep:	12327-500
Tel. Comercial:	21 2654-8697
Tel. Residêncial:	21 3879-6123
Tel. Contato:	21 9854-2348

Editar Cadastro

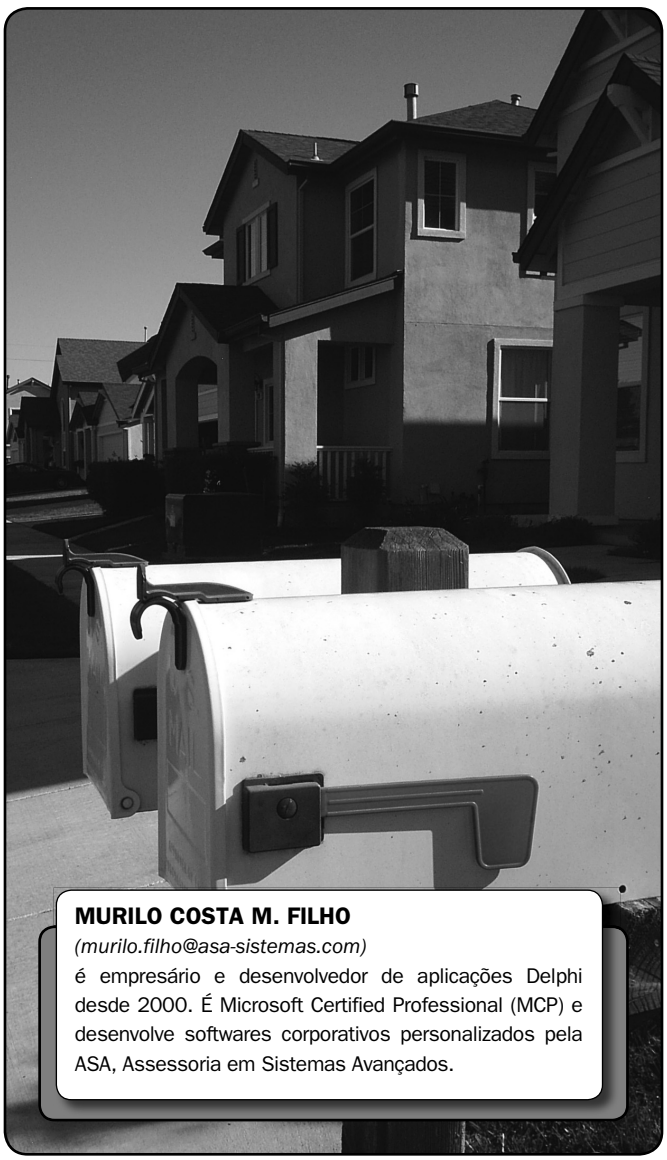
Assinaturas Ativas
ClubeDelphi - Revistas Enviadas
MSDN Magazine - Revistas Enviadas

DevMedia GROUP

Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

www.devmedia.com.br/central

Envio automatizado de emails com Delphi e Outlook



MURILO COSTA M. FILHO
(murilo.filho@asa-sistemas.com)
é empresário e desenvolvedor de aplicações Delphi desde 2000. É Microsoft Certified Professional (MCP) e desenvolve softwares corporativos personalizados pela ASA, Assessoria em Sistemas Avançados.

Recentemente fui convidado a incluir um recurso em um aplicativo, para que ele fosse capaz de gerar, automaticamente, e-mails através do Outlook. A idéia era percorrer todos os registros da tabela de funcionários, e para cada funcionário cadastrado, gerar um e-mail, através de um *template* previamente configurado pelo usuário, com os dados da mala direta.

Entre outros recursos, o aplicativo fornecia ao usuário a opção de gerar mensagens tanto em formato texto como em HTML. Neste artigo apenas demonstrarei recursos básicos, fornecendo uma introdução ao tema. Em outros artigos abordaremos recursos mais complexos, como, por exemplo, anexar arquivos, importar lista de contatos (ou exportar) e aplicação de filtros em mensagens recebidas.

No final da série, desenvolveremos um aplicativo completo que controlará o envio e recebimento de e-mails, em um banco de dados de rastreabilidade de mensagens, com o uso do *Turbo Power OfficePartner* (*freeware*).

O aplicativo é bem simples, mas as idéias podem ser aplicadas em diversos outros projetos, como por exemplo, envio personalizado de e-mails para clientes aniversariantes, alerta de produtos com baixo estoque etc. Basta usar a criatividade.

Introdução sobre automação

Automação, como o próprio nome sugere, significa automatizar ou controlar algo. No conceito da programação, significa controlar uma aplicação através de outra. Funciona como um contrato entre as duas aplicações, e esse (contrato) funciona através de interfaces.

No Delphi, utilizamos as *Type Libraries*, que é uma espécie de repositório onde são expostas as funções disponíveis de uma aplicação. Neste artigo utilizaremos o Delphi 7, mas o exemplo pode ser aplicado nas versões posteriores (Delphi 2005 e 2006).

O Delphi 7 já vem com as *Type Libraries* importadas do Office 97, 2000 e XP. No caso trabalharemos com o Office 2003 e utilizaremos a unit *OutlookXP*. O exemplo pode ser aplicado com o Office XP.

Um exemplo simples

Crie um novo projeto no Delphi, e na seção *uses* adicione as units *OutlookXP*, *OleServer* e *ComObj*. No evento *OnCreate* do formulário adicione o código da **Listagem 1**.

Listagem 1. Exemplo simples de envio de e-mail

```
var
  NmSpace: NameSpace;
  Outlook: TOutlookApplication;
  MI: MailItem;
begin
  Outlook := TOutlookApplication.Create(nil);
  Outlook.ConnectKind := ckRunningOrNew;
  Outlook.Connect;
  try
    NmSpace := Outlook.GetNamespace('MAPI');
    NmSpace.Logon('', '', False, False);
    MI := Outlook.CreateItem(olMailItem) as MailItem;
    MI.Recipients.Add('destinatario@provedor.com.br');
    MI.Subject := 'Testando assunto';
    MI.Body := 'Testando corpo da mensagem';
    MI.Display(False);
  finally
   try
     nmSpace.LogOff;
   except
   end;
   try
     Outlook.Disconnect;
   except
   end;
end;
```

O código da listagem anterior é bem simples. Declaramos a variável *Outlook* do tipo *TOutlookApplication* para "conectarmos" ao Outlook. Não é necessário configurar a propriedade *AutoConnect* para *True*, pois o Outlook requer o Login em um perfil MAPI para que seja possível o envio e recebimento de e-mails.

Esse processo foi realizado através da variável *NmSpace*. A variável *MI* do tipo *MailItem* é a variável que contém as propriedades e métodos para compor a mensagem. As principais propriedades são:

- *Recipients*: Coleção de destinatários, onde utiliza o *Add (const Name:Widestring)* para adicionar os endereços;
- *Subject*: Assunto da mensagem;
- *Body*: Corpo da mensagem para e-mail somente texto;
- *HtmlBody*: Corpo da mensagem para e-mail HTML;

Ao executar o projeto, recebemos uma mensagem do Outlook, semelhante a **Figura 1**.

A caixa de diálogo é exibida por questões de segurança pelo Outlook quando um aplicativo externo tenta controlá-lo. Sempre que o aplicativo gerar uma nova mensagem, o Outlook solicitará ao usuário qual o tempo de permissão para acesso ao aplicativo, o que é incômodo e nada prático, uma vez que queremos automatizar o processo de criação de mensagens.

A única maneira de contornar esse problema é utilizando uma biblioteca chamada *Redemption*, que abordaremos a seguir.

A biblioteca Redemption

RDO ou *Redemption Objects* é uma biblioteca desenvolvida por *Dmitry Streblechenko* para facilitar o acesso a determinadas propriedades e funções "escondidas" pelo modelo de objetos do Outlook. Entre elas, está a limitação imposta pelo Patch de Segurança do Outlook. A biblioteca é *freeware* e está disponível em: *www.dimastr.com/redemption/Redemption.zip*.

Para instalar, execute o arquivo *Install.exe*. Após, é necessário importar a *Type Library* da biblioteca no Delphi 7, através do menu *Project>Import Type Library*. Clique no botão *Add* e aponte para o arquivo *Redemption.dll* (em *System32* ou na pasta de instalação da biblioteca). Clique em *Create Unit* (**Figura 2**).

Uma nova unit será criada, chamada *Redempetion_TLB.pas* na pasta *Imports* do Delphi.

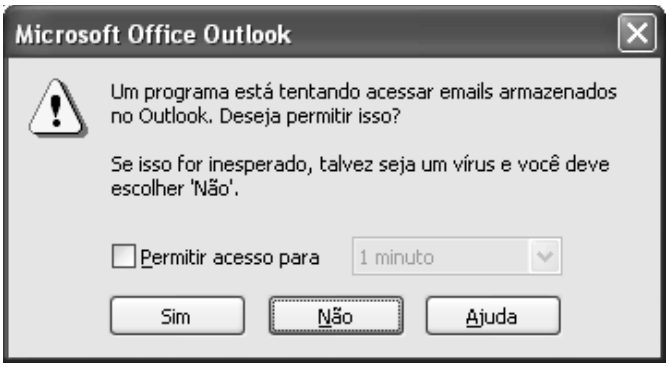


Figura 1. Mensagem de segurança do Outlook

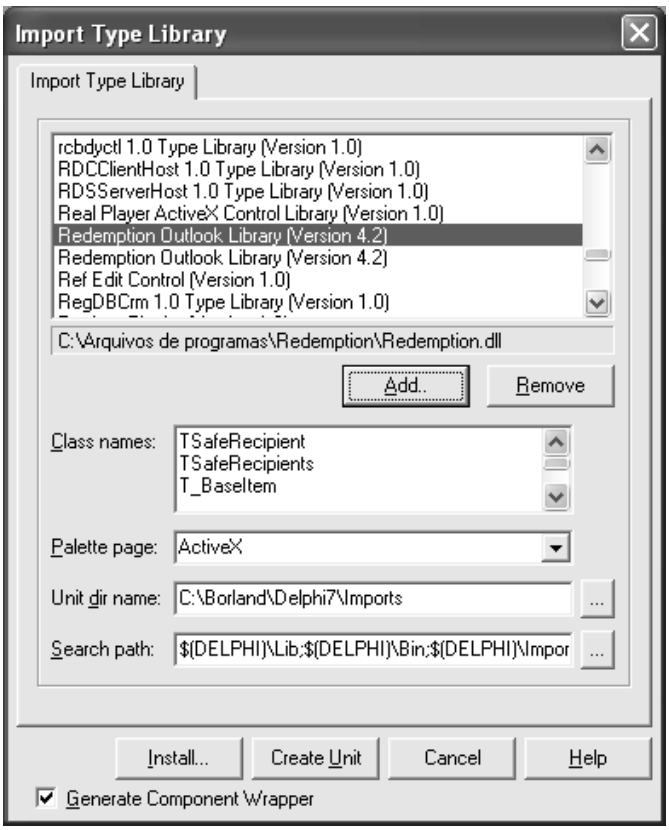


Figura 2. Tela para importação e criação de Type Libraries (Delphi 7)

Alterando o código para utilizar a biblioteca

Utilizaremos o mesmo código inicial para gerarmos uma mensagem no Outlook, sem a exibição da caixa de diálogo de segurança, utilizando o código da **Listagem 2**.

Listagem 2. Alterando o código para usar a biblioteca Redempetion

```
var
  NmSpace: NameSpace;
  MAPIFolder: OleVariant;
  sFolder: OleVariant;
  Outlook : TOutlookApplication;
  MI, oItem : OleVariant;
begin
  Outlook := TOutlookApplication.Create(nil);
  Outlook.ConnectKind := ckRunningOrNew;
  Outlook.Connect;
  try
    NmSpace := Outlook.GetNamespace('MAPI');
    NmSpace.Logon('', '', False, False);
    oItem := Outlook.CreateItem(0);
    MI := CreateOleObject('Redemption.SafeMailItem');
    MI.Item := oItem;
    MI.Recipients.Add('destinatario@provedor.com.br');
    MI.Subject := Software para automação de envio de '+
      'mensagens via Outlook';
    MI.Body := 'Testando mensagem de emails';
    MAPIFolder := Outlook.Session.GetDefaultFolder(olFolderInbox);
    sFolder := CreateOleObject('Redemption.MAPIFolder');
    sFolder.Item := MAPIFolder;
    sFolder.Display;
    MI.Display;
  finally
   try
     nmSpace.LogOff;
   except
   end;
   try
     Outlook.Disconnect;
   except
   end;
  end;
end;
```

A variável *MI*, para composição da mensagem, foi substituída pelo tipo *Variant*, mas as propriedades e métodos são os mesmos. Uma nova variável, *oItem*, também do tipo *Variant*, foi criada para armazenar o objeto OLE *Redempention.SafeMailItem*, que permite o acesso ao objeto *SafeMailItem*, oculto no Outlook, que "pula" a exibição da tela de diálogo de segurança.

A variável *MAPIFolder* recebe a pasta *Inbox* (Caixa de Entrada) que é exibida após a criação da mensagem, através da variável *sFolder* e do método *Display*. Se rodar a aplicação, verá que o Outlook será aberto, uma nova mensagem será criada com os valores colocados no código (para o destinatário, assunto e corpo do e-mail).

Exemplo de criação automatizada de mensagens

Alteraremos o código anterior para gerar e enviar, automaticamente, as mensagens para todos os funcionários de uma determinada tabela. Para o exemplo, utilizarei o banco *Northwind*, disponível no Access ou SQL Server 2000 (MSDE), que é distribuído com o Office.

É possível utilizar o mesmo código para qualquer banco de dados. O único requisito principal é de que a tabela possua um campo para armazenamento do endereço de e-mail dos destinatários. Abra o *Northwind.mdb* (localizado em *<instalação>\OFFICE11\SAMPLES\Northwind.mdb*), edite a tabela *Funcionários* e adicione o campo "Email", do tipo *Texto* com tamanho "150" (**Figura 3**). Salve e edite os dados, adicionando os e-mails de cada funcionário.

No Delphi, crie um novo projeto e adicione um *ADOConnection* e um *ADODataSet*. No *ADOConnection*, configure a conexão para o banco *Northwind* (propriedade *ConnectionString*). Altere a propriedade *LoginPrompt* para *False* e dê o nome de "Database". No *ADODataSet*, altere a propriedade *Connection* para *Database* e *CommandText* para o seguinte código:

```
SELECT EmployeeId, TitleofCourtesy, LastName,
FirstName, Email
FROM Employees
```

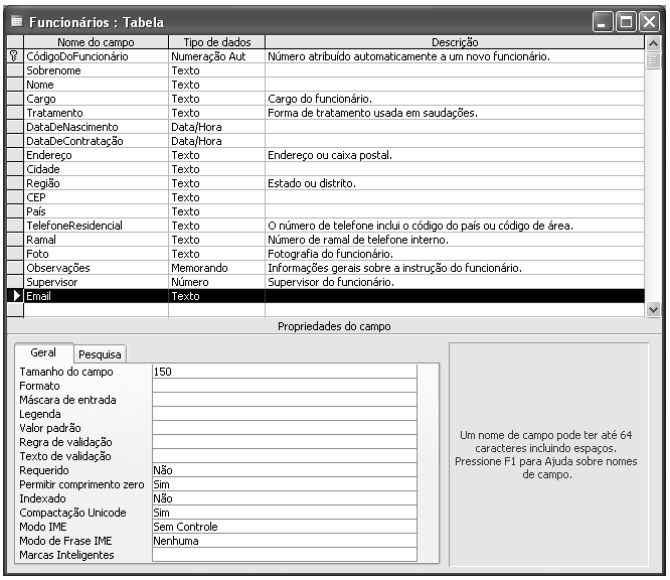


Figura 3. Editando a tabela de Funcionários

Figura 4. Tela principal do aplicativo

Adicione um *Label*, um *Memo*, e dois *Buttons*. Configure-os como na **Figura 4** (adicione um texto no *Memo* que será o corpo do e-mail, adicionando os campos do *DataSet* entre colchetes).

A idéia é bem simples. Os campos do banco de dados ficam entre chaves ([]) e o aplicativo substituirá os campos pelos dados, gerando as mensagens para todos os funcionários cadastrados. O usuário digita o texto desejado para a mala direta e clica em *Enviar Emails*, para iniciar o processo. No *uses* adicione as seguintes units:

```
uses OleServer, OutlookXP, Redemption_TLB, ComObj;
```

O código dos eventos *OnCreate* do formulário, *AfterOpen* do *ADODataSet* e botão *Limpar Texto* encontra-se na **Listagem 3**

Listagem 3. Código do OnCreate, AfterPost e botão

```
procedure TfrmPrincipal.FormCreate(Sender: TObject);
begin
 dsFunc.Open;
end;

procedure TfrmPrincipal.dsFuncAfterOpen(
  DataSet: TDataSet);
begin
 if (DataSet.State <> dsInactive) then
  lblFunc.Caption := IntToStr(DataSet.RecordCount) +
    ' funcionário(s) encontrado(s)';
end;

procedure TfrmPrincipal.btnLimparClick(
  Sender: TObject);
begin
 memTexto.Clear;
end;
```

Declare na seção *private* a seguinte função:

```
function SubstituirCampoDados(Linha, Campo, Texto: string): string;
```

Implemente-a, com o código da **Listagem 4.**

Figura 5. Outlook exibindo os itens enviados

Listagem 4. Função SubstituiCamposDados

```
{ Função para substituir variáveis por dados da tabela }
function TfrmPrincipal.SubstituirCampoDados(Linha,
  Campo, Texto: string): string;
var
 sPos, sLen: Integer;
 sTexto, sCampo, sLinha: string;
begin
 sTexto := Texto;
 sCampo := Campo;
 sLinha := Linha;

 sPos := pos(sCampo, sLinha);
 sLen := length(sCampo);

 Delete(sLinha, sPos, sLen);
 Insert(sTexto, sLinha, sPos);

 Result := sLinha;
end;
```

SubstituirCampoDados substitui as variáveis entre chaves (ex.: [NOME]) pelo valor correspondente ao campo na tabela. Para fi-

“ O Maior Portal para Desenvolvedores da América Latina.”

Acesse:

www.devmedia.com.br

DevMedia GROUP

nalizar, no botão *Enviar Emails*, adicione o código da **Listagem 5**.

Ao clicar em *Enviar Email*, o aplicativo conecta ao Outlook, percorre todos os registros da tabela, gerando uma mensagem para cada registro, e após gerar as mensagens, abre o Outlook exibindo o progresso do envio dos e-mails (**Figura 5**).

> **Observação:** É normal as mensagens ficarem salvas temporariamente na pasta Rascunhos, ao invés da Caixa de Saída. As mensagens serão transferidas para a Itens Enviados após serem transmitidas.

Conclusão

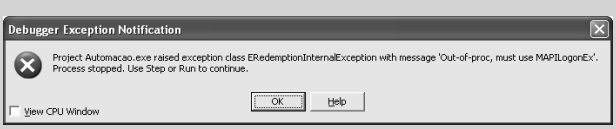
O exemplo desenvolvido é bem básico, constituindo-se numa introdução ao tema. Muitos recursos podem ser incluídos, que abordaremos no futuro, como uma barra de progresso de envio para cada e-mail (no caso de tabelas com muitos registros), a possibilidade de incluir arquivos em anexo e também a composição de mensagens em formato HTML.

Recomendo a utilização de *Threads* para envio em massa de registros ou o Outlook poderá não responder a todas as solicitações. Além disso, cabe ressaltar que vários outros aplicativos podem ser "controlados" a partir do Delphi, como o *Lotus Notes*, caso a empresa não utilize o Outlook.

Basta pesquisar as funções de cada aplicativo e aplicá-las no Delphi, após importar as *Type Libraries* correspondentes. Bons estudos e até o próximo artigo. ■

Mensagem de erro

Quando você executar o projeto dentro do IDE do Delphi, será mostrada uma mensagem de erro, semelhante a figura a seguir:



Trata-se de uma exceção interna da classe ERedemptionInternal Exception, que não altera a funcionalidade da automação, ou seja, os e-mails são enviados normalmente. Consultei o autor da biblioteca, Dmitry Streblechenko, e fui instruído a ignorar o erro quando compilado através do IDE, alegando que, nas próximas versões da biblioteca, a exceção não será mais exibida.

Links

Biblioteca Redemption

www.dimastr.com/redemption

Dicas de automação em Delphi

delphi.about.com/od/kbcontrolole/OLE_Automation_using_Delphi.htm

www.djpate.freeserve.co.uk/Automation.htm

Dicas de automação Outlook via Delphi

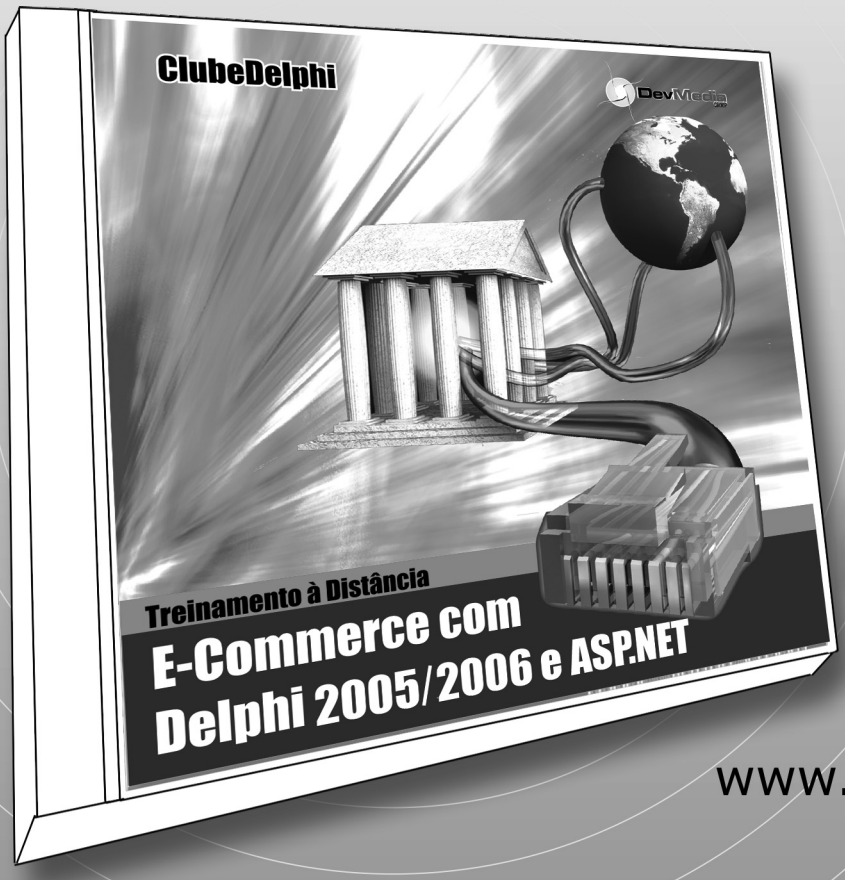
www.djpate.freeserve.co.uk/AutoOutl.htm

Listagem 5. Enviando os e-mails

```
procedure TfrmPrincipal.btnEnviarClick(Sender: TObject);
var
 Outlook: TOutlookApplication;
 NmSpace: NameSpace;
 MI, oItem, MAPIFolder, sFolder: OleVariant;
 TextoEmail: TStrings;
 sAssunto, sCampo, sDados: string;
 CorpoEmail: WideString;
 a, b, c: integer;
begin
 { Iniciando conexão com o Microsoft Outlook }
 lblInfo.Caption := 'Conectando ao Outlook...';
 Application.ProcessMessages;
 try
   Outlook := TOutlookApplication.Create(nil);
   Outlook.ConnectKind := ckRunningOrNew;
   Outlook.Connect;
 except
   lblInfo.Caption := 'Erro ao conectar ao Outlook...';
   Application.ProcessMessages;
   Abort;
 end;
 try
   lblInfo.Caption := 'Buscando perfil de acesso...';
   Application.ProcessMessages;
   NmSpace := Outlook.GetNamespace('MAPI');
   NmSpace.Logon('', '', False, False);
   TextoEmail := TStringList.Create;
   sAssunto := edtAssunto.Text;
   { Para cada registro, enviar um e-mail }
   lblInfo.Caption := 'Percorrendo registro de funcionários...';
   Application.ProcessMessages;
   while not dsFunc.EOF do
   begin
     lblInfo.Caption := 'Gerando mensagem para: ' +
       dsFunc.FieldByName('NOME').AsString;
     Application.ProcessMessages;

     { Limpando as variáveis }
     CorpoEmail := '';
     TextoEmail.Clear;
     TextoEmail.Assign(memTexto.Lines);

     { Localizando e substituindo variáveis entre chaves [] por dados }
     for a := 0 to TextoEmail.Count - 1 do
       for b:= 0 to dsFunc.Fields.Count-1 do
       begin
         sCampo := Uppercase(dsFunc.Fields[b].FieldName);
         sDados := dsFunc.FieldByName(sCampo).AsString;
         if pos('['+ sCampo +']', TextoEmail[a]) > 0 then
           TextoEmail[a] := SubstituirCampoTexto(
             TextoEmail[a], '['+ sCampo +']', sDados);
       end;

     { Iniciando criação da mensagem }
     oItem := Outlook.CreateItem(0);
     MI := CreateOleObject('Redemption.SafeMailItem');
     MI.Item := oItem;
     MI.Item.BodyFormat := olFormatPlain;
     MI.Recipients.Add(dsFunc.FieldByName('EMAIL').AsString);
     MI.Subject := sAssunto;
     { Armazenando a mensagem na variável temporária
       CorpoEmail, do tipo Widestring }
     for c := 0 to TextoEmail.Count - 1 do
       CorpoEmail := CorpoEmail + #13 + .Strings[c];
     { Concluindo criação da mensagem e movendo ao
       próximo registro }
     MI.Body := CorpoEmail;
     MI.Send;
     dsFunc.Next;
   end;

   { Exibir o Outlook e progresso do envio de mensagens }
   lblInfo.Caption := 'Concluindo criação de mensagens...';
   Application.ProcessMessages;
   MAPIFolder := Outlook.Session.GetDefaultFolder(olFolderInbox);
   sFolder := CreateOleObject('Redemption.MAPIFolder');
   sFolder.Item := MAPIFolder;
   sFolder.Display;
 finally
  lblInfo.Caption := 'Limpando variáveis...';
  Application.ProcessMessages;
  try
    nmSpace.LogOff;
  except
  end;
  try
    Outlook.Disconnect;
  except
  end;
 end;
 lblInfo.Caption := 'Operação concluída com sucesso...';
 Application.ProcessMessages;
end;
```

Treinamentos a distância.

Aprenda de forma prática, eficiente e econômica.



E-Commerce com Delphi 2005/2006 e ASP.NET

Aprenda a desenvolver poderosas aplicações Web através do Delphi 2005/2006 e as tecnologias ASP.NE WebBroker, Intraweb e WebSnap.
O curso conta com módulos textuais;
35 horas de vídeo-aulas;
fórum exclusivo para os alunos e suporte on-line.

COMPRE JÁ O SEU!

www.devmedia.com.br/curso

Outros Curso

Delphi e Relatórios

Com este curso você desenvolverá relatórios através do Delphi 7, do Delphi 2005 e Delphi 2006, utilizando as ferramentas Rave Reports, Quick Reports e Crystal Report. Além do módulo textual o curso contém mais de 20 horas de vídeo-aulas; fórum exclusivo e suporte

Evento à distância - Introdução ao .NET

Conheça os recursos do ASP.NET com o Visual Studio 2003 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a IDE do VS; componentização; DataGrid e criar aplicações de BD com o ADO.NET! São 6h de palestras em formato de vídeo-aulas e você pode tirar suas dúvidas com os palestrantes!

Evento à distância - Novidades do Delphi 2005

Conheça os recursos do Delphi 2005 e aprenda a criar páginas ASP.NET; Webservices; trabalhar com a nova IDE do Delphi; utilizar programação OO no Delphi 2005; trabalhar em equipe com o novo Starteam e criar aplicações de BD com o ADO.NET!

DevMedia

Pensando simples com a classe TApplication

HENRIQUE GUSSO NETZKA
(henrique@vetorsistemas.com.br)
é atualmente analista/programador na sua própria empresa, a GHN - Soluções Inteligentes. Está na área de informática há 12 anos, sendo 5 anos profissionalmente. Atua hoje com Business Intelligence e desenvolvimento para automação comercial em Curitiba/PR.

Ao desenvolvermos uma aplicação completa, acabamos deixando de lado, muitas vezes por falta de tempo, alguns detalhes que facilitariam a vida do usuário e, certamente, a do suporte técnico e equipe de desenvolvimento também.

É muito comum, por exemplo, surgir a necessidade de visualizarmos a tela do usuário no momento em que aparece um erro, uma mensagem "estranha" a ele, ou mesmo para explicar ao usuário uma situação que ele não consegue entender.

Um dos recursos mais comuns para esse tipo de situação é pedir ao usuário que ele faça um *Print Screen* da tela e envie-o ao suporte técnico ou à pessoa responsável. O que apresentarei aqui, então, é uma forma simples de salvar esse *Snapshot* diretamente para um arquivo (do tipo *Bitmap*), sem que ele precise abrir um editor de imagens ou "colar" a imagem dentro de um documento e salvar o mesmo.

Com dois componentes e algumas linhas de código, exibiremos uma caixa de diálogo ao usuário e salvaremos o *Snapshot* no arquivo indicado tão logo ele pressione a tecla *Print Screen*.

Outro ponto negativo para os usuários do Delphi 7 ou anterior, é a incompatibilidade dos softwares desenvolvidos com o novo formato dos arquivos de ajuda da Microsoft, os HTML Helps (CHM). Esse impasse obriga, muitas vezes, a documentar o sistema nos famosos *WinHelps* (HLP), que são antigos, visualmente pesados e obsoletos.

Com algumas linhas de código, porém, vamos inverter essa situação, fazendo com que a aplicação interaja normalmente com um arquivo no formato HTML Help. E por último, teremos um tratamento de exceções personalizado.

Afinal, quem nunca teve problemas em entender o que o usuário tentava dizer quando estourou na sua tela uma famosa violação de acesso em algum endereço de memória.

Neste artigo, demonstrarei como substituir a mensagem da exceção por uma mensagem amigável, e em paralelo fazer um *log* da exceção com informações técnicas que ajudarão na resolução do problema.

Inserindo e configurando os componentes na aplicação

Crie uma nova aplicação no Delphi e adicione um *ApplicationEvents* (*Additional*) e um *SaveDialog* (*Dialogs*). Usaremos o *ApplicationEvents* para sermos notificados quando a aplicação gerar alguns eventos de nosso interesse, e o *SaveDialog* servirá para darmos ao usuário a opção de escolher onde o arquivo do *Print Screen* será gravado.

Configure o *SaveDialog* com as seguintes propriedades:

- *Title* digite "Salvando Print Screen...";
- *DefaultExt* digite "*.bmp"
- *Filter* adicione: "Bitmaps (*.bmp)|*.bmp".

Essas configurações garantirão que o usuário saberá do que se trata a caixa de diálogo quando ela for exibida. Como o arquivo gerado será necessariamente do tipo Bitmap (BMP), deixamos essa como única opção de filtro (*Filter*).

Interceptando a tecla Print Screen

Selecione o *ApplicationEvents* e no evento *OnMessage*, digite o código da **Listagem 1**.

Perceba que a primeira verificação é conferir se a mensagem é do tipo que esperamos, no nosso caso, quando o usuário "soltar" uma tecla (*WM_KeyUp*), e essa for a *Print Screen* (*VK_SNAPSHOT*). Isso é importante, pois o evento *OnMessage* recebe centenas de notificações durante a execução da aplicação.

Tendo a certeza de que estamos reagindo ao evento correto, chamamos a caixa de diálogo para salvar o arquivo. Após o usuário informar o nome do arquivo e clicar em *Salvar*, extraímos a imagem da tela do usuário e a gravamos num arquivo com formato *Bitmap*.

Para extrair a imagem da tela do usuário, associamos um *Canvas* (*CanvasAux*) à própria imagem do desktop. Ou seja: temos uma espécie de "porta aberta", dentro da nossa aplicação, para a imagem exibida na tela.

As dimensões dessa imagem, portanto, serão as dimensões da resolução do usuário. Por isso, atribuímos ao *RectAux* (*TRect*), bem como ao *Bmp* (*TBitmap*), os limites (largura e altura) iguais à largura da tela (*Screen.Width*) e à altura da tela (*Screen.Height*), respectivamente.

Com a imagem do desktop disponível e as novas dimensões configuradas, fazemos então a cópia daquela imagem para dentro do *Bitmap* (variável *Bmp*). Para essa cópia, utilizamos o *CopyRect*, que fará literalmente uma cópia do *CanvasAux* (*desktop*) para o *Canvas* do *Bmp*.

Com o BMP preenchido, basta gravá-lo para o arquivo informado pelo usuário. Para isso, utilize o *SaveToFile* da classe *TBitmap*. E pronto! Seu *Snapshot* estará gravado em um arquivo sem o uso de qualquer editor de imagens ou similares.

JPEG: Economia e praticidade

Agora que você já sabe como capturar a imagem da tela e colocá-la num BMP, vamos implementar uma nova funcionalidade: converteremos o BMP para JPEG. A principal vantagem que teremos ao fazer isso, é que os arquivos JPEG são estrondosamente menores, o que facilitará o trânsito desse dentro de uma rede ou via Web. Para convertê-lo, crie um método chama "BMPToJPEG", como mostra a **Listagem 2**.

Coloque a chamada a esse código, logo acima do método da **Listagem 1**, e nesse método inclua o código da **Listagem 3**.

Repare que adicionamos duas novas linhas, uma para criar e outra para destruir o novo objeto JPEG, e substituímos a linha *Bmp.SaveToFile* pela conversão para JPEG seguida da gravação a partir do novo formato. É importante lembrar que o *SaveDialog* deve estar configurado para um arquivo JPG, e não mais com extensão BMP.

Considerações importantes

Você deve ter percebido um segundo parâmetro no evento *OnMessage*, a *Handled*. Ele indicará à aplicação se ela deve parar ali ou continuar processando aquele evento. Caso você inclua no código: *Handled := True*, você informará à aplicação que você já "cuidou" daquela mensagem, e que a aplicação não precisa mais "se preocupar" com ela.

Listagem 1. Respondendo a tecla Print Screen

```
procedure TForm1.ApplicationEvents1Message(
  var Msg: tagMSG; var Handled: Boolean);
var
  Bmp: TBitmap;
  CanvasAux: TCanvas;
  RectAux: TRect;
begin
  if (Msg.message = WM_KeyUp) and
    (Msg.wParam = VK_SNAPSHOT) then
  begin
    Bmp := TBitmap.Create;
    CanvasAux := TCanvas.Create;
    try
      CanvasAux.Handle := GetWindowDC(GetDesktopWindow);
      RectAux := Rect(0, 0, Screen.Width, Screen.Height);
      Bmp.Width := Screen.Width;
      Bmp.Height := Screen.Height;
      Bmp.Canvas.CopyRect(RectAux, CanvasAux, RectAux);
      if SaveDialog1.Execute then
        Bmp.SaveToFile(SaveDialog1.FileName);
    finally
      Bmp.Free;
      CanvasAux.Free;
    end;
  end;
```

Listagem 2. Convertendo de BMP para JPEG

```
uses
  JPEG;
...
procedure BMPToJPEG(ABitmap: TBitmap; var AJPEG: TJPEGImage);
begin
  AJPEG.CompressionQuality := 100;
  AJPEG.Assign(ABitmap);
end;
```

Listagem 3. Salvando o arquivo em JPEG

```
var
  JPEG: TJPEGImage;
begin
...
   Bmp := TBitmap.Create;
   { Adicione essa linha aqui }
   JPEG := TJPEGImage.Create;
   ...
   //Bmp.SaveToFile(SaveDialog1.FileName);
   BMPToJPEG(Bmp, JPEG);
   JPEG.SaveToFile(SaveDialog1.FileName);
 finally
   ...
   { Lembre-se de dar um Free no novo objeto }
   JPEG.Free;
 end;
```

No nosso caso, especificamente, a diferença será que o *Print Screen* do usuário **não** será colocado na área de transferência (comportamento padrão do Windows quando a tecla *Print Screen* é pressionada), será apenas gravado no arquivo que o usuário escolher, por isso adicione o código anterior no evento.

HTML Help sem complicações

Logo acima da seção *implementation* da unit, inclua a seguinte linha:

```
function HtmlHelp(hwndCaller: THandle; pszFile: PChar;
  uCommand: cardinal; dwData: longint): THandle;
  stdcall; external 'hhctrl.ocx' name 'HtmlHelpA';
```

O que fizemos com o código anterior, foi simplesmente declarar uma *procedure* externa à nossa aplicação, no caso, ela está alocada no arquivo *hhctrl.ocx*, um arquivo de sistema do Windows. Altere a propriedade *KeyPreview* do formulário para *True*. Declare agora uma constante referente ao nome do seu arquivo de help, dentro da seção *implementation*:

```
const
  ArquivoDeAjuda = 'C:\MeuHelp.chm';
```

> **Dica:** Para aprender sobre a criação de arquivos de help e integração com o Delphi, consulte as edições 52 e 68.

Em *OnMessage* do *ApplicationEvents* (a mesma utilizada na **Listagem 1**), inclua o código da **Listagem 4**, que será responsável por chamar o arquivo de ajuda, utilizando-se da *procedure* externa (declarada anteriormente) toda vez que a tecla F1 for pressionada na sua aplicação.

Agora, selecione o componente novamente e no evento *OnHelp* digite o código da **Listagem 5**.

Na listagem anterior temos uma outra novidade: os *Contexts*. Um arquivo de ajuda do Windows pode ser indexado de várias formas, sendo que uma delas é através de contextos (que são valores numéricos ligados a tópicos de ajuda). Eles estão presentes, por exemplo, quando você chama uma caixa de diálogo com a função *MessageDlg* (é o último parâmetro).

Caso você pressione F1 numa caixa de diálogo com *HelpContext* definido, o código anterior será acionado e o arquivo de help será aberto no ponto que o programador definiu (por exemplo, no tópico onde a mensagem é explicada!). Caso o mesmo não tenha sido definido, a página inicial do help será exibida ao usuário.

Reconhecendo os HelpContexts dos seus formulários

Para que essa funcionalidade fique mais profissional, uma boa medida é configurar os *HelpContexts* dos seus formulários para valores correspondentes dentro do arquivo de help. Porém, de nada adiantará configurar as propriedades se você não implementar a função que chama o Help para considerar esses valores.

Nesse sentido, vamos modificar a **Listagem 4**, fazendo com que ela chame o Help informando a página desejada, conforme o código da **Listagem 6**.

A partir desse momento, caso o seu formulário ativo possua um *HelpContext* válido, o Help será exibido diretamente na página referente a ele. Observe que mantivemos a compatibilidade com os formulários que não possuem a propriedade configurada, nesse caso, o Help terá sua página inicial exibida.

Quando declaramos *HtmlHelp*, passamos a interagir diretamente com a API do Windows, portanto, pode ser uma boa idéia procurar por explicações e documentação sobre tal função no próprio site da Microsoft, ao invés de procurar apenas em sites de Delphi.

Indo mais longe: controlando as exceções exibidas ao usuário

Procure pelo evento *OnException* no *ApplicationEvents* e digite o código da **Listagem 7**.

Listagem 4. Respondendo ao botão F1

```
procedure TForm1.ApplicationEvents1Message(
  var Msg: tagMSG; var Handled: Boolean);
begin
  ... { Código do Print Screen }
  if (Msg.message = WM_KEYDOWN) and (Msg.wParam = VK_F1) then
  begin
    Handled := True;
    HtmlHelp(Application.Handle, PChar(ArquivoDeAjuda), $F, 0);
  end;
end;
```

Listagem 5. O evento OnHelp do ApplicationEvents

```
function TForm1.ApplicationEvents1Help(Command: Word;
  Data: Integer; var CallHelp: Boolean): Boolean;
begin
  CallHelp := True;
  if (Command = $F) then
    HtmlHelp(Application.Handle, PChar(ArquivoDeAjuda), $F, Data)
  else
    HtmlHelp(Application.Handle, PChar(ArquivoDeAjuda), $F, 0);
end;
```

Listagem 6. Interpretando a propriedade HelpContext do formulário ativo

```
procedure TForm1.ApplicationEvents1Message(
  var Msg: tagMSG; var Handled: Boolean);
begin
  ...
  Handled := True;
  if Assigned(Screen.ActiveForm) then
    HtmlHelp(Application.Handle, PChar(ArquivoDeAjuda), $F,
      Screen.ActiveForm.HelpContext)
  else
    HtmlHelp(Application.Handle, PChar(ArquivoDeAjuda), $F, 0);
  ...
end;
```

Listagem 7. Tratando as exceções da aplicação

```
procedure TForm1.ApplicationEvents1Exception(
  Sender: TObject; E: Exception);
var
  ArqLog: TStringList;
begin
  if (E is EAbort) then
    Exit;
  ArqLog := TStringList.Create;
  try
    if FileExists('C:\Erros.log') then
      ArqLog.LoadFromFile('C:\Erros.log');
    ArqLog.Add('** ERRO **');
    ArqLog.Add('Data: ' + DateTimeToStr(Now));
    ArqLog.Add('Mensagem: ' + E.Message);
    ArqLog.Add('');
    ArqLog.SaveToFile('C:\Erros.log');
    MessageDlg('Houve um erro na aplicação e um log '+
      'de erros foi gerado.'#13#10#13#10'Contate o '+
      'suporte técnico imediatamente!', mtInformation, [mbOK], 0);
  finally
    ArqLog.Free;
  end;
end;
```

A partir desse momento, todas as exceções geradas pela aplicação passarão por esse evento, ao invés de serem exibidas ao usuário. Devemos lembrar, porém, que o comando *Abort* gera uma exceção silenciosa, da classe *EAbort*. Por isso, antes de mais nada, ignoramos tal tipo de exceção para que o usuário não receba nenhuma mensagem estranha quando utilizarmos um *Abort* no código.

Tendo ignorado a exceção silenciosa, bem como outras exceções que poderão ter sua exibição ao usuário desejada, vamos ao tratamento. O primeiro passo será criar um *StringList*, uma interface simples e rápida que permite, entre outras coisas, manipular arquivos de texto-puro de forma extremamente fácil.

Para que um log de erros eventualmente existente não seja sobreposto, verificamos a sua existência e carregamos seu conteúdo para o *StringList*, fazendo com que o mesmo insira as novas informações após o final do arquivo já existente.

Após termos completado o *StringList* e salvado para o arquivo novamente, exibimos uma mensagem amigável ao usuário, orientando-o sobre os procedimentos a serem tomados e, ao mesmo tempo, guardando informações para que possamos tratar o erro posteriormente. Veja na **Figura 1**, a mensagem e o arquivo de log.

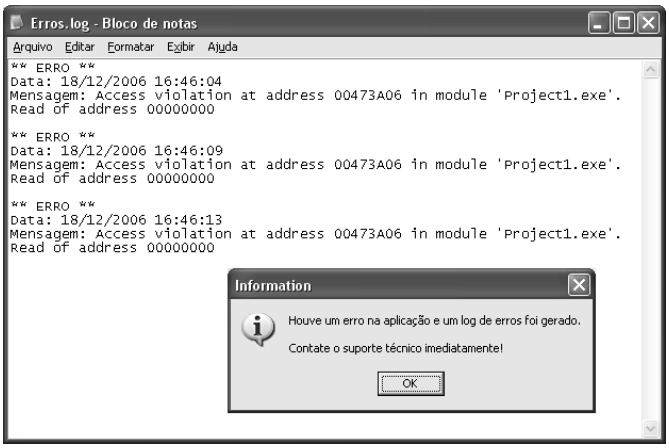


Figura 1. Exemplo salvamento de log

Conclusão

Existem muitas formas de melhorar as propostas apresentadas neste artigo, e a idéia é que elas realmente sejam melhoradas. O que mostrei aqui foi uma pequena luz, um pequeno início, mostrando funcionalidades que muitas vezes parecem obscuras ou distantes da nossa realidade.

Para o *Print Screen*, por exemplo, você poderia criar sua própria caixa de diálogo para salvar o *Snapshot*, de forma a deixá-la mais amigável e auto explicativa. No caso do Help, use e abuse dos *Contexts* nos formulários. Lembre-se que facilitar o acesso à ajuda do sistema pode diminuir as ligações de usuários para o seu suporte técnico em busca de sanar dúvidas simples.

E para as exceções, melhorias não faltam. Pense, por exemplo, que você pode registrar informações de usuário logado na hora do erro, tela ativa (*Screen.ActiveForm*), controle ativo (*Screen.ActiveControl*) e até mesmo lista de telas abertas (*Screen.Forms[]*).

Indo ainda mais longe, é possível implementar uma rotina que busque pelo *Call Stack* e complemente as informações gravadas no *log*. E, por último, com todas as informações reunidas, o sistema poderia gerar e enviar um e-mail diretamente ao suporte técnico contendo as informações e indicando que houve um erro na aplicação. ■

PENSE...

QUANTO TEMPO
VOCÊ GASTARIA
PARA DESENVOLVER
COBRANÇA COM BOLETOS
BANCÁRIOS PARA
APENAS UM BANCO
NO SEU SOFTWARE

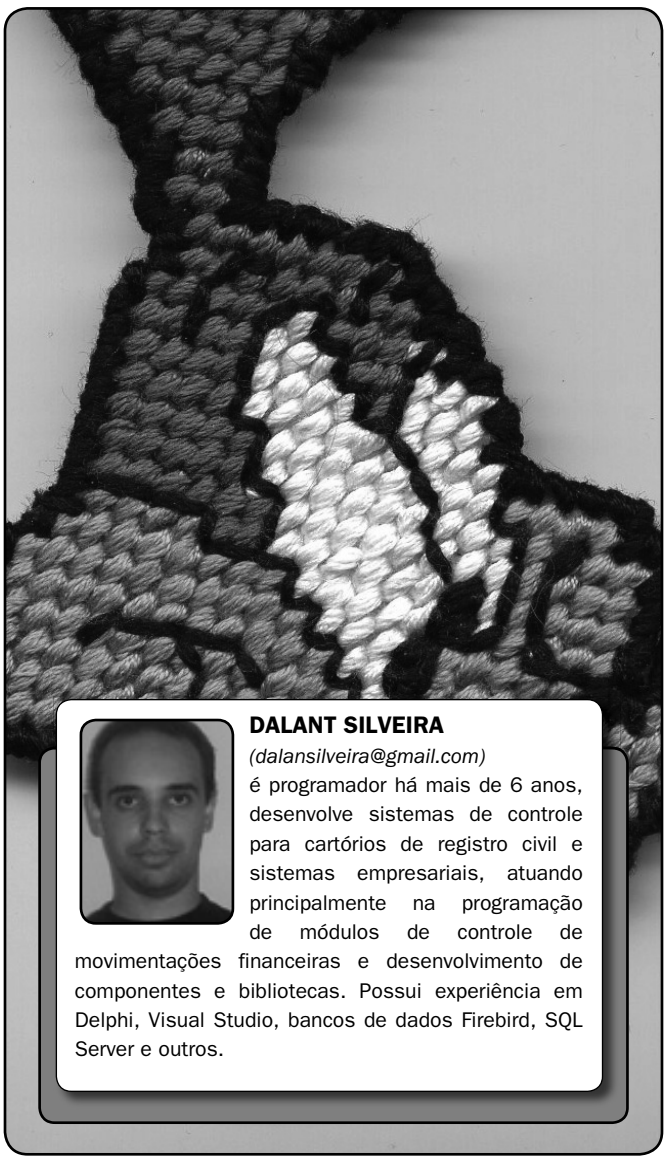
COBREBEMX

- 56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
- GERAÇÃO DE BOLETOS ON LINE;
- GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
- MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

Cobre Bem Tecnologia

DOWNLOADS E INFORMAÇÕES EM WWW.COBREBEM.COM

Integrando sua aplicação com a busca de CEPS dos Correios – Win32



DALANT SILVEIRA
(dalansilveira@gmail.com)
é programador há mais de 6 anos, desenvolve sistemas de controle para cartórios de registro civil e sistemas empresariais, atuando principalmente na programação de módulos de controle de movimentações financeiras e desenvolvimento de componentes e bibliotecas. Possui experiência em Delphi, Visual Studio, bancos de dados Firebird, SQL Server e outros.

O dia a dia da empresa não deixa tempo para nada, é preciso fazer mais coisas em menos tempo, e o preenchimento de cadastros muitas vezes chega a ser irritante e repetitivo. Esse é um desafio para o programador, adivinhar o que o usuário precisará digitar é impossível, mas podemos ajudá-lo usando ferramentas e nossa própria imaginação para criar artifícios que os ajudem a serem mais produtivos.

Mostrarei uma forma de reduzir bastante o tempo e trabalho de inclusão de um cadastro de fornecedor, mostrando na prática como utilizar o Web Service de consulta de CEPs dos Correios, onde o usuário digitará o CEP do fornecedor e o endereço completo será preenchido automaticamente através da busca de logradouros, diminuindo drasticamente a quantidade de campos a serem preenchidos.

O Web Service de CEPs já existe a um bom tempo, porém com um pequeno detalhe, ele foi criado apenas para ser integrado ao Office 2003, dificultando um pouco o trabalho de integração com outros aplicativos, apesar disso é perfeitamente possível utilizá-lo a partir de aplicações Delphi.

Isso ficou um pouco obscuro durante um tempo, até que finalmente o nosso amigo Rodrigo Sendin trilhou o caminho das pedras e em seu artigo na edição anterior mostrou como realizar o processo em uma aplicação Delphi for .NET. De forma semelhante, transportei a idéia para o mundo Win32 e resolvi então escrever este artigo para demonstrar aos usuários de Delphi (VCL) como acessar o serviço e usá-lo de uma forma útil.

Importando o Web Service dos Correios

Podemos encontrar no site dos Correios (*www.correios.com.br*) quase no rodapé da página principal, o sistema de busca de CEPs. No mesmo quadro, mais abaixo encontramos um link que direciona para "outras formas de consulta" (*www.correios.com.br/servicos/cep/cep_loc_log.cfm*).

Nessa página, no menu lateral esquerdo existe o link: *Veja como consultar o CEP no Office da Microsoft* (*www.correios.com.br/parcerias/cep/office2003/default.cfm*), onde você encontra as instruções de como instalar o Web Service no Office 2003.

Seguindo as instruções da página, durante o processo de instalação, o Office exibe uma janela com um endereço direcionando para o Web Service (**Figura 1**), a extensão ASMX indica que o Web Service foi criado com ASP.NET, e será esse endereço que usaremos para fazer as consultas.

Criando o projeto

Crie um novo projeto no Delphi. Para obter acesso ao Web Service usaremos SOAP através de uma interface que o Delphi gerará por meio do *WSDL Importer*. No menu *File>New>Other*, selecione a aba *WebServices* e então o *WSDL Importer*.

No *wizard* que aparecerá, digite o endereço do Web Service. O valor a ser digitado, é: "http://consultacep.correios.com.br/office2003/Query.asmx?WSDL". O *?WSDL* serve para que o Web Service retorne sua descrição e funções remotas disponíveis, se quiser ver o retorno detalhado abra o endereço anterior em seu navegador (**Figura 2**).

Depois de digitar o endereço, clique em *Next* e veja que o Delphi gerou uma unit com a interface de acesso ao Web Service, que possui apenas uma função chamada *Query* (**Figura 3**).

Após finalizar o assistente você terá uma nova unit em seu projeto, *Query.pas*. Como nosso objetivo é criar um cadastro de fornecedores, precisaremos de um banco de dados e a devida tabela para armazenar os fornecedores cadastrados. Neste exemplo utilizaremos o Firebird como banco de dados e os componentes dbExpress para acessá-lo.

Criando o banco de dados

Criei uma estrutura simples para a tabela de fornecedores, apenas para demonstrar o funcionamento do Web Service. Vamos então criar o banco de dados de testes através do *script* da **Listagem 1,** usando seu gerenciador Firebird preferido ou mesmo usando o *isql* na pasta *bin* do Firebird. Lembre-se de criar a pasta mostrada no *script* ou altere o caminho do arquivo para o de sua preferência.

Listagem 1. Script para criação do banco de dados de exemplo

```
CREATE DATABASE 'LOCALHOST:C:\CLUBEDELPHI\TESTE_CEP.FDB'
USER 'SYSDBA' PASSWORD 'masterkey' PAGE_SIZE 4096;

CREATE TABLE FORNECEDORES (
  ID INTEGER NOT NULL,
  RAZAO_SOCIAL VARCHAR(200) NOT NULL,
  CEP VARCHAR(8) NOT NULL,
  ENDERECO VARCHAR(200),
  NUMERO INTEGER,
  BAIRRO VARCHAR(100),
  CIDADE VARCHAR(150),
  UF CHAR(2),
  CNPJ_CPF VARCHAR(18),
  TELEFONE VARCHAR(22),
  CONTATO VARCHAR(200),
  WEBSITE VARCHAR(200),
  EMAIL VARCHAR(200));

COMMIT WORK;
ALTER TABLE FORNECEDORES ADD PRIMARY KEY (ID);
COMMIT WORK;
```

Figura 1. Extraindo o endereço do Web Service do Office

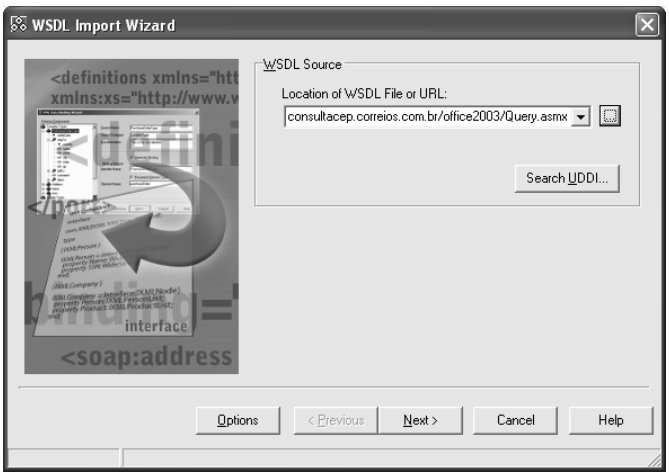


Figura 2. WSDL Import do Delphi

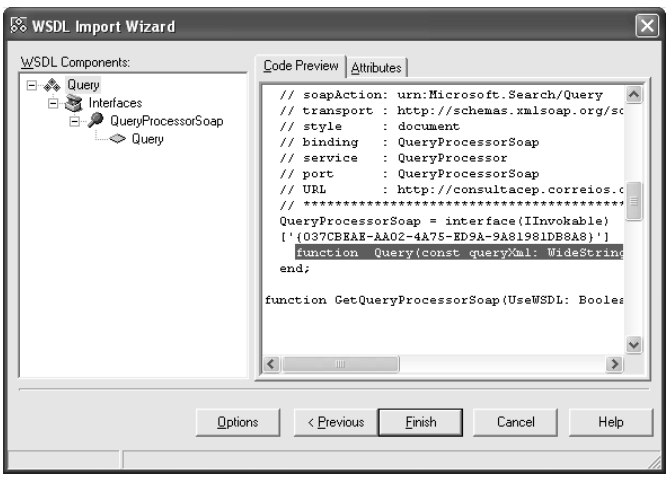


Figura 3. Interface gerada pelo WSDL Import para acesso ao Web Service

Criando o cadastro de fornecedores

Após criar o banco de dados, volte ao Delphi e adicione ao formulário os componentes: *SQLConnection*, *SimpleDataset*, *DataSource*, um *DBNavigator* e configure-os como descrito a seguir:

- SQLConnection
  - Name: "SQLCon"
  - ConnectionName: "IBConnection"
  - LoginPrompt: *False*
  - Params[SQLDialect]: "3"
  - Params[Database]: "C:\CLUBEDELPHI\TESTE_CEP.FDB "
- SimpleDataSet
  - Name: "sdstFORNECEDORES"
  - Connection: *SQLCon*
  - Dataset>CommandText: "select * from FORNECEDORES"
- DataSource
  - Name: "dsFORNECEDORES"
  - DataSet: *sdstFORNECEDORES*
- DBNavigator
  - Name: "dbNav"
  - DataSource: *dsFORNECEDORES*

Adicione *DBEdits* e botões ao formulário, nomeie-os como descrito na **Figura 4**, ligue os *DBEdits* ao *DataSource* e está pronta nossa tela de cadastro, podemos agora partir para a utilização do Web Service.

> **Nota:** Para esse exemplo simples, não usaremos um Data Module e nem o trio de componentes *SQLDataSet + DataSetProvider + ClientDataSet*.

Busca pelo CEP

Com nosso cadastro de fornecedores funcionando, vamos a integração com o Web Service de consulta de logradouros. O primeiro passo é adicionar a unit *Query.pas* ao *uses* do formulário de cadastro, depois vamos analisar a função *Query* conforme apresentada no código a seguir:

```
QueryProcessorSoap = interface(IInvokable)
['{037CBEAE-AA02-4A75-ED9A-9A81981DB8A8}']
  function Query(const queryXml: WideString): WideString; stdcall;
end;
```

Veja que a função precisa de um parâmetro: *queryXml* do tipo *WideString*, para podermos enviar nosso parâmetro de pesquisa, no caso o CEP digitado, é preciso formatá-lo em um XML. Outro detalhe é que a pesquisa retorna um outro *WideString*, que também é um XML. Então além de enviarmos nossa consulta através de um XML, teremos que fazer o *parse* da *string* retornada para exibir seu resultado nos campos do formulário.

Para fazer esse *parse* usaremos a interface *IXMLDocument*. Adicione no *uses* do formulário as units: *XmlDoc*, *XmlIntf*, *XmlDom* e na seção *private* o seguinte código:

```
function GeraXMLQuery(Cep: string): string;
```

Na **Listagem 2** temos a implementação da função.

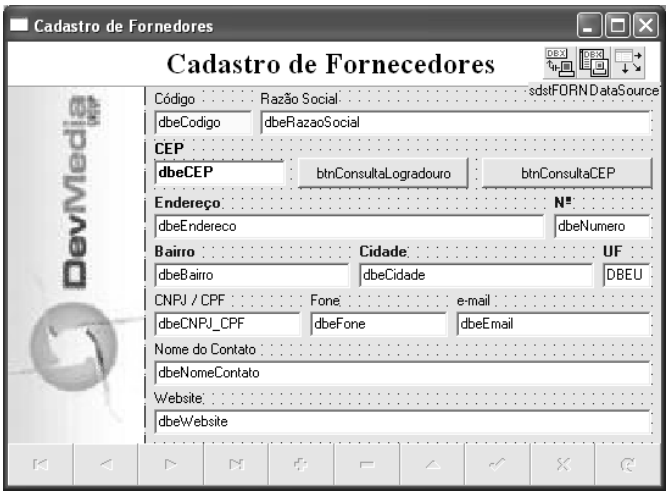


Figura 4. Tela de cadastro de fornecedores

Listagem 2. Função que gera o XML de consulta de Logradouros

```
function TFrmCadFornecedores.GeraXMLQuery(
  Cep: string): string;
begin
  Result := '<?xml version="1.0" encoding="utf-8"?>'+#13#10+
    '<QueryPacket xmlns="urn:Microsoft.Search.Query">'+ #13#10+
    '  <Query>'+#13#10+
    '    <Context>'+#13#10+
    '      <Context>'+#13#10+
    '        <QueryText>'+Cep+'</QueryText>'+#13#10+
    '      </Context>'+#13#10+
    '      <OfficeContext xmlns="urn:Microsoft.Search.Query.'+
    '       Office.Context">'+#13#10+
    '        <ApplicationContext>'+#13#10+
    '          <Name>Microsoft Office</Name>'+#13#10+
    '          <Version>(11.0.6568)</Version>'+#13#10+
    '        </ApplicationContext>'+#13#10+
    '      </OfficeContext>'+#13#10+
    '    </Context>'+#13#10+
    '  </Query>'+#13#10+
    '</QueryPacket>';
end;
```

Analisando o código anterior note que o XML é gerado a partir de concatenação de texto, eu poderia usar o *IXMLDocument* (interface para o *XMLDocument*) que acompanha o Delphi, para gerar o XML de saída, mas por ser um XML simples, resolvi fazê-lo dessa forma.

Assim também fica mais claro como será enviada a consulta, esse *schema* é padrão para os serviços de pesquisa do Office. Você pode encontrar mais sobre os serviços de consulta do Office através do link: *msdn2.microsoft.com/en-us/library/aa193595(office.11).aspx*. Estudando um pouco o funcionamento você poderá até mesmo criar seus próprios Web Services para integração com o Office.

Vamos implementar também duas funções: uma auxiliar, chamada *Split*, e outra que fará o retorno do Web Service, implementadas na **Listagem 3**.

No código anterior, fazemos o *parse* do XML retornado, varremos os elementos em busca da tag <*Data*>. Para ler o seu conteúdo precisaremos seguir a ordem: *ResponsePacket>Response>Range>Results>Content>Actions* e por fim *Data*. Veja na **Listagem 4** o modelo de um XML de retorno do Web Service.

Listagem 3. Função de processamento do resultado da consulta

```
function Split(const separator: string; const str: string):
TStringList;
begin
  Result := TStringList.Create;
  Result.Text := StringReplace(str, separator, #13#10, [rfReplaceAll]);
end;

procedure TFrmCadFornecedores.ProcessaRetornoQueryLogradouro(
 XMLRetorno: String);
var
  XmlDoc: IXMLDocument;
  Node: IXMLNode;
  DOMDataNode: IDOMNode;
  AuxStrings: TStringList;
  AuxString: string;
begin
  { Verifica se a pesquisa retornou algum resultado }
  if Pos('não retornou nenhum resultado', XMLRetorno) > 0 then

  begin
    Application.MessageBox('Nenhum logradouro encontrado!',
      'Consulta...', MB_OK + MB_ICONINFORMATION);
    Exit;
  end;

  { Verifica se o serviço está disponível }
  if Pos('Falha ao acessar o sistema', XMLRetorno) > 0 then

  begin
    Application.MessageBox('O serviço está indisponível!',
      'Consulta...', MB_OK + MB_ICONINFORMATION);
    Exit;
  end;

  { Verifica se o DataSet está em modo de inclusão/edição }
  if not(sdstFORNECEDORES.State in [dsEdit, dsInsert]) then
    Exit;
    { Carrega o XML no Parser }
  XmlDoc := LoadXMLData(XMLRetorno);

  try
    XmlDoc.Active := True;
    XmlDoc := LoadXMLData(FormatXMLData(XmlDoc.XML.Text));
    XmlDoc.SaveToFile(ExtractFilePath(
      Application.ExeName) + 'XMLRETORNO.XML');
    { Localiza o node Results }

    try
      Node := XmlDoc.DocumentElement.ChildNodes.FindNode('Response').
        ChildNodes.FindNode('Range').ChildNodes.FindNode('Results');
        { vai até o Content/Actions }
      DomDataNode := Node.DOMNode.childNodes.item[
        0]{content}.childNodes[6]{actions};
      { Define os valores dos campos com o valores
         encontrados na busca pelo CEP }
       AuxString := DomDataNode.childNodes[2]{data}.lastChild.nodeValue;
       AuxString := StringReplace(AuxString,'<<COMPLEMENTO>>', '', []);
       AuxStrings := Split(',', AuxString);

       try
         sdstFORNECEDORES.FieldByName(
           'ENDERECO').AsString := Trim(AuxStrings[0]);
       finally
         AuxStrings.Free;
       end;

      sdstFORNECEDORES.FieldByName('BAIRRO').AsString := DomDataNode.childNodes[
         3]{data}.lastChild.nodeValue;
         { cidade }
       AuxString := DomDataNode.childNodes[4]{data}.lastChild.nodeValue;
         AuxStrings := Split('-', AuxString);
       try
        sdstFORNECEDORES.FieldByName('CIDADE').AsString := Trim(AuxStrings[0]);
        sdstFORNECEDORES.FieldByName('UF').AsString := Trim(AuxStrings[1]);
       finally
         AuxStrings.Free;
       end;

    except
      on E: Exception do
      begin
        MessageBox(Handle, PChar('Não foi possível' +
        'interpretar o resultado da consulta!' +
        #13#10 + E.Message), '', MB_ICONWARNING or MB_OK);
      end;
    end;
      XmlDoc.Active := False;

 finally
   XmlDoc := nil;
 end;

end;
```

Nessa estrutura os elementos <*Data*> não têm distinção um de outro, reconheceremos os mesmos pela sua ordem, na seguinte seqüência:

- Node Data Nº 1 – Endereço
- Node Data Nº 2 – Bairro
- Node Data Nº 3 – Cidade
- Node Data Nº 4 – CEP

Listagem 4. XML retornado na pesquisa de endereços

```
<?xml version="1.0"?>
<ResponsePacket revision="1" xmlns="urn:Microsoft.Search.Response">
<Response domain="{2698075D-E2F5-4254-87B2-7FC9E9AB0780}">
<QueryId>{8E823A74-053D-4434-A67B-D1CD050B3349}</QueryId>

<Range>
  <Results>
     <Content xmlns="urn:Microsoft.Search.Response.Content">
      <Char bold="true">A sua busca retornou o seguinte resultado:</Char>
      <HorizontalRule/>
      <P>Avenida Brasil, 99999</P>
      <P>Centro</P>
      <P>Belo Horizonte - MG</P>
      <P>99999-999</P>
      <Actions>

          <Insert>
            <Text>Inserir Endereço</Text>
           </Insert>

           <Copy>
            <Text>Copiar Endereço</Text><
          /Copy>

           <Data>AVENIDA BRASIL, 99999 &lt;&lt;COMPLEMENTO&gt;&gt;</Data>

           <Data>CENTRO</Data>

           <Data>BELO HORIZONTE - MG</Data>

           <Data>99999-999</Data>

      </Actions>

   <HorizontalRule/>
   <P>
     <Char bold="true">Fornecido por</Char>
   </P>
   <Image source="http://consultacep.correios.com.br/office2003/
                  Imagens/logo.gif"/>
   <P>
     <Char bold="true">www.CORREIOS.com.br</Char>
   </P>

   <HorizontalRule/>
   <P>Atendimento ao Cliente</P>
   <P>0800 57 00100</P>

   </Content>
  </Results>
</Range>

 <Status>SUCCESS</Status>

</Response>

</ResponsePacket>
```

Listagem 5. Código do botão que faz a requisição

```
procedure TFrmCadFornecedores.btnConsultaLogradouroClick(Sender: TObject);
var
  qry: QueryProcessorSoap;
  XMLQuery,
  XMLRetorno: string;
begin
  if Length(dbeCEP.Text) < 8 then
    Exit;
  qry := GetQueryProcessorSoap();
  { Gera o XML com CEP para efetuar a pesquisa }
  XMLQuery := GeraXMLQuery(dbeCEP.Text);
  { Recebe o XML de retorno }
  XMLRetorno := qry.Query(XMLQuery);
  { Faz o processamento do XML preenchendo os campos se o retorno
    encontrou o logradouro relacionado ao CEP que você digitou }
  ProcessaRetornoQueryLogradouro(XMLRetorno);
end;
```

No *OnClick* do botão de consulta adicione o código da **Listagem 5**, onde vamos gerar o XML de consulta e enviar ao Web Service, que por sua vez devolverá o resultado que será processado pelo *parser IXMLDocument*.

Nesse ponto, o aplicativo já está funcional, faça um teste, aperte F9, digite apenas o CEP e clique no botão. Teremos o endereço, bairro, cidade e estado preenchidos automaticamente, evitando a digitação de quatro campos. Faremos agora o processo inverso, buscar o CEP a partir do endereço digitado.

Busca pelo Logradouro

Vimos como buscar o endereço apenas digitando o CEP, agora vou mostrar como fazer o inverso, buscar o número do CEP pelo endereço digitado. Adicione ao formulário um *ListView*,alterando a propriedade *Visible* para *False* e adicione três colunas: "CEP", "Bairro" e "Endereço". Altere a propriedade *ViewStyle* para *vsReport*.

Mostraremos o *ListView* apenas quando clicarmos no *btnConsultaCEP*. Troque o nome do *ListView* para "lvCEPs" e então adicione o código da **Listagem 6**, para implementação da função *GeraXMLQueryCEP*.

Semelhante a *GeraXMLQuery* nossa função de busca de CEPs também é uma concatenação de texto. Precisamos enviar ao Web Service o endereço que procuramos o CEP. Note que ao invés de enviarmos o CEP na tag <*Query*>, enviamos o texto *Logradouro.*

A seguir, enviamos os outros dados como parâmetros da pesquisa, concluímos então que ao enviar o texto *Logradouro* o Web Service entende que a pesquisa é uma pesquisa de CEP e procura nos parâmetros os dados para a consulta.

Ao enviar esse XML para o Web Service ele devolverá novamente um XML, teremos então que interpretá-lo e exibir os resultados no *ListView*. Adicione ao botão de consulta o código da **Listagem 7**.

No código anterior testamos se os campos foram preenchidos corretamente, geramos o XML com a pesquisa através *GeraXmlQueryCEP* e passamos o XML gerado para a função *ProcessaRetornoQueryCEP*. Nela fazemos o *parse* do XML retornado através da interface *IXMLDocument*, lendo cada nível do XML até chegar a tag <*Data*>.

> **Nota:** Existe a restrição que os campos devem ser preenchidos, mas você pode buscar pelo Logradouro e Cidade, mas note que a quantidade de registros pode ser muito grande.

Adicione também o código da **Listagem 8** que preenche os valores de retorno no *ListView*.

Nesse caso a estrutura do XML é pouco diferente, temos uma sequência maior de *nodes*: *ResponsePacket>Response>Range>Results>Content*. Nesse ponto teremos tantos elementos <*Heading*> quantos forem os resultados retornados, pois é possível haver mais de um CEP para a mesma rua/avenida etc.

Precisamos fazer um loop pelos resultados e neste exemplo passar cada um deles para uma linha no *ListView* que foi criada anteriormente. Fazemos então o *loop* pelos nós a partir do elemento

Listagem 6. Função que gera o XML de pesquisa de CEPs

```
function TFrmCadFornecedores.GeraXMLQueryCEP(UF,
  Cidade, TipoLogradouro, Logradouro,
  Numero: string): string;
begin
  Result :=
    '<?xml version="1.0" encoding="utf-8"?>'+#13#10+
    '<QueryPacket xmlns="urn:Microsoft.Search.Query">'+#13#10+
    '  <Query>'+#13#10+
    '    <Context>'+#13#10+
    '      <QueryText>Logradouro</QueryText>'+#13#10+
    '      <Requery>'+#13#10+
    '        <ServiceParameters xmlns="urn:Microsoft.Search.'+
    '          Office.ServiceParameters">'+#13#10+
    '          <Parameters>'+#13#10+
    '            <AdvancedSearchEditUF>'+UF+
    '              </AdvancedSearchEditUF>'+#13#10+
    '            <AdvancedSearchEditLocalidade>'+Cidade+
    '            </AdvancedSearchEditLocalidade>'+#13#10+
    '            <AdvancedSearchEditTipoLogradouro>'+
    '             TipoLogradouro+'</AdvancedSearchEditTipoLogradouro>'+#13#10+
    '            <AdvancedSearchEditLogradouro>'+Logradouro+
    '            </AdvancedSearchEditLogradouro>'+#13#10+
    '            <AdvancedSearchEditNumero>'+Numero+
    '            </AdvancedSearchEditNumero>'+#13#10+
    '          </Parameters>'+#13#10+
    '        </ServiceParameters>'+#13#10+
    '      </Requery>'+#13#10+
    '    </Context>'+#13#10+
    '    <OfficeContext xmlns="urn:Microsoft.Search.Query.'+
    '     Office.Context">'+#13#10+
    '      <ApplicationContext>'+#13#10+
    '        <Name>Microsoft Office</Name>'+#13#10+
    '        <Version>(11.0.6568)</Version>'+#13#10+
    '      </ApplicationContext>'+#13#10+
    '    </OfficeContext>'+#13#10+
    '  </Query>'+#13#10+
    '</QueryPacket>';
end;
```

Listagem 7. Código do botão de consulta de CEPs

```
procedure TFrmCadFornecedores.btnConsultaCEPClick(Sender: TObject);
var
  qry: QueryProcessorSoap;
  XMLQuery, XMLRetorno, AuxString,TipoLogradouro: String;
  AuxStrings: TStringList;
  I: Integer;
begin
  if lvCEPs.Visible = True then
  begin
    lvCEPs.Visible := False;
    Exit;
  end;
  lvCEPs.Visible := True;
  lvCEPs.BringToFront;
  { Validação de preenchimento dos campos }
  if Length(dbeEndereco.Text) = 0 then
    Exit;
  if Length(dbeNumero.Text) = 0 then
    Exit;
  if Length(dbeBairro.Text) = 0 then
    Exit;
  if Length(dbeCidade.Text)= 0 then
    Exit;
  if Length(dbeUF.Text) <> 2 then
    Exit;
  { Prepara a busca }
  qry := GetQueryProcessorSoap();
  AuxString := dbeEndereco.Text;
  AuxStrings := Split(' ', AuxString);
  try
    if AuxStrings.Count < 2 then
      Exit;
    TipoLogradouro := Trim(AuxStrings[0]);
    AuxString := '';
    for I := 1 to AuxStrings.Count -1 do
      AuxString := AuxStrings[I];
    XMLQuery := GeraXMLQueryCEP(dbeUF.Text,
      dbeCidade.Text, TipoLogradouro, AuxString,dbeNumero.Text);
  finally
    AuxStrings.Free;
  end;
  { recebe o XML de retorno }
   XMLRetorno := qry.Query(XMLQuery);
  { Faz o processamento do XML preenchendo os campos se o retorno
    encontrou o logradouro relacionado ao CEP que você digitou }
   ProcessaRetornoQueryCEP(XMLRetorno);
end;
```

Listagem 8. Função que faz o parse da consulta de CEPs

```
procedure TFrmCadFornecedores.ProcessaRetornoQueryCEP(
 XMLRetorno: String);
var
  XmlDoc: IXMLDocument;
  Node: IXMLNode;
  DOMDataNode: IDOMNode;
  AuxItem: TListItem;
  I: Integer;
begin

  { Verifica se a pesquisa retornou algum resultado }
  if Pos('não encontrou nenhum resultado', XMLRetorno) > 0 then
  begin
    Application.MessageBox('Nenhum CEP encontrado!',
      'Consulta...', MB_OK + MB_ICONINFORMATION);
    Exit;
  end;

  { Verifica se o serviço está disponível }
  if Pos('Falha ao acessar o sistema', XMLRetorno) > 0 then
  begin
    Application.MessageBox(
      'O serviço está indisponível!',
      'Consulta...', MB_OK + MB_ICONINFORMATION);
    Exit;
  end;

  XmlDoc := LoadXMLData(XMLRetorno);
  try
    XmlDoc.Active := True;
    XmlDoc := LoadXMLData(FormatXMLData(XmlDoc.XML.Text));
    XmlDoc.SaveToFile(ExtractFilePath(
      Application.ExeName) + 'XMLRETORNOCEP.XML');
    lvCEPs.Clear;
    { Localiza o node Results }
    try
      Node := XmlDoc.DocumentElement.ChildNodes.FindNode('Response').
        ChildNodes.FindNode('Range').ChildNodes.FindNode('Results');
        { Vai até o Content }
      DomDataNode := Node.DOMNode.childNodes.item[
        1]{Content};
      { faz um loop pelos resultados }
      for I := 0 to
        DomDataNode.childNodes.Length -1 do
      begin
        if DomDataNode.childNodes[I].nodeName = 'Heading' then
        begin
          AuxItem := lvCEPs.Items.Add;
          { CEP }
          AuxItem.Caption := DomDataNode.childNodes[I].
            childNodes[4]{Actions}.childNodes[5]{Data}.
            lastChild.nodeValue;
          { Bairro }
          AuxItem.SubItems.Add(DomDataNode.childNodes[
            I].childNodes[4]{Actions}.childNodes[3]{Data}.
            lastChild.nodeValue);
          { Endereço }
          AuxItem.SubItems.Add(DomDataNode.childNodes[
            I].childNodes[4]{Actions}.childNodes[2]{Data}.
            lastChild.nodeValue);
        end;
      end;
      Self.Update;
    except
      on E: Exception do
      begin
        MessageBox(Handle,
          PChar('Não foi possível interpretar' +
          ' o resultado da consulta!' + #13#10 +
          E.Message), '', MB_ICONWARNING or MB_OK);
      end;
    end;
      XmlDoc.Active := False;
  finally
    XmlDoc := nil;
  end;
end;
```

Links

Site dos Correios

www.correios.com.br

Serviços de pesquisa do Office

msdn2.microsoft.com/en-us/library/aa193595(office.11).aspx

Figura 5. Pesquisando CEP a partir do logradouro

<*Content*> e testamos se o nome do *node* é igual a *Heading*, se sim, adicionamos os *subnodes* contidos no *node* seguindo a ordem:

- Node Data Nº 2 – Endereço + Número
- Node Data Nº 3 – Bairro
- Node Data Nº 5 – Número do CEP

Dica: Coloque sempre o tipo de Logradouro (Rua, Avenida, Alameda etc.) antes do nome do mesmo. Uma alternativa é criar um campo tipo de logradouro, sendo que o XML de retorno do Web Service contém os tipos corretos. Se utilizar essa funcionalidade, é necessário alterar o código do botão, pois nele estamos pegando o tipo de logradouro (usando o *Split*).

Conclusão

Vimos como integrar a nossa aplicação ao Web Service de CEPs dos Correios através de uma simulação, fazendo nossa aplicação interagir como o Office e interpretar o Web Service, fazendo o *parse* de um XML complexo usando a interface *IXMLDocument*. Usamos essas tecnologias para diminuir o trabalho do digitador em uma tela de cadastros.

Existem diversas outras situações onde esse Web Service poderá ser útil, como por exemplo, para determinar se o CEP digitado corresponde ao endereço, validando assim o CEP e reduzindo erros, neste exemplo apenas demonstrei uma forma prática de utilização desse recurso.

Sempre digo que quanto mais você puder simplificar o trabalho de quem usa seu software melhor ele será. Na informática as pessoas tendem a gostar de inovações, principalmente se essas inovações ajudam a cumprir determinadas tarefas. O bom programador é aquele que não tem preguiça de inovar ou mesmo de copiar certas idéias, tirando uma coisa daqui e outra dali, você surpreenderá a si mesmo e àqueles que usarem seus softwares.

Espero que todos tenham gostado e que tirem bom proveito do que aqui foi ensinado, até a próxima. ■

Crie passo a passo um leitor de RSS com o Delphi

MICHAEL BENFORD
(michael@devmedia.com.br)
é Acadêmico do curso de matemática da Universidade Federal Fluminense e desenvolvedor Delphi e .NET da QSi, Qualidade em Sistemas de Informação. Programa em Delphi há 5 anos, desenvolvendo aplicações cliente/servidor, multicamadas, Web, utilitários de uso geral, componentes, experts e aplicativos não-comerciais. É colunista do portal da DevMedia, articulista e membro da comissão editorial da revista ClubeDelphi. Palestrante da primeira edição do Developer's Webdays e da BorCon Brasil 2006.

Informação sempre foi algo precioso, e obtê-la nunca foi fácil. Com o advento da Internet e sua popularização, o acesso à informação foi enormemente facilitado. Mas, como todos sabemos, existe muita (muita mesmo!) coisa na Internet, e garimpar o que interessa nem sempre é uma tarefa simples, e por que não dizer, confortável de fazer.

Não seria muito melhor se fosse possível escolher exatamente o assunto desejado e esperar que ele venha até nós, e não o contrário? Bom, como provavelmente você leitor sabe, isso não é nenhuma previsão minha. Algo assim já existe e está em plena expansão, atendendo pelo nome de sindicalização, traduzido de *Syndication*. E nesse meio, o formato que mais se destaca atualmente é o RSS.

Uma breve história do formato RSS

Embora pareça novidade, o RSS existe desde 1999. A primeira versão foi criada por *Ramanathan V. Guha*, na época funcionário da *Netscape*. Desde então o formato foi aprimorado por várias pessoas diferentes, e hoje temos a versão 2.0 (internamente, 2.0.1), implementada em 2002 e declarada como "congelada" pelo atual proprietário do formato, a Universidade de Harvard (Fonte: Wikipedia).

Isso significa, segundo a especificação oficial, que "nenhuma mudança significativa pode ser feita e que futuros trabalhos devem ser realizados sob outro nome". Porém, ambas as versões 1.0 e 2.0 do RSS são ainda largamente utilizadas por muitos Web Sites e grandes empresas, o que indica que o formato dificilmente desaparecerá em curto prazo.

O acrônimo RSS varia de significado, de acordo com a versão referida:

- *RDF Site Summary* (0.9 e 1.0);
- *Rich Site Summary* (0.91 e 1.0);
- *Really Simple Syndication* (2.0).

Dica: Para conhecer mais sobre esse formato, consulte a seção *Links*.

Milhares de sites oferecem notícias no formato RSS. Geralmente, para acessar o conteúdo publicado basta apontar o seu browser para uma URL mais ou menos assim: *www.site.com.br/noticias.xml*. Esse tipo de link também é conhecido como *Web Feed* ou apenas *feed*.

Nesse artigo vamos aprender como consumir o conteúdo desses *feeds* criando um projeto no Delphi e, "de quebra", aprender várias coisas sobre programação com a ferramenta.

Como funciona o RSS

O formato RSS é baseado na linguagem XML, estruturado em *tags* que representam o conteúdo sendo distribuído. Essas tags estão definidas na especificação oficial (veja seção *Links*). Portanto, consumir informações no formato RSS não é nada mais do que ser capaz de ler um arquivo XML. Veja na **Listagem 1** um exemplo de documento RSS.

Vamos ver o que algumas dessas tags significam:

- *rss*: É a tag raiz (*root*) de um documento RSS. Junto dela vem o atributo *version*, para informar qual versão do formato está sendo utilizada;
- *channel*: Canal de notícias. Dentro dessa tag ficam o conteúdo do RSS e informações extras;
  - *title*: Título do canal;
  - *link*: URL para acessar o canal;
  - *description*: Descrição do conteúdo oferecido pelo canal;
  - *image*: Imagem que representa o canal. Na maioria das vezes é o logotipo do site de informações produtor do RSS;
  - *item*: Representa uma notícia dentro do XML. Pode haver várias tags desse tipo dentro do arquivo.

Você pode encontrar todas as demais tags e o que cada uma significa através da especificação do formato RSS (seção *Links*). Deixando a teoria de lado, começaremos nosso projeto a partir do próximo tópico.

Criando o formulário principal

Para esse artigo eu utilizei o Turbo Delphi Professional, mas você poderá seguir o exemplo usando outras versões. As diferenças, quando existirem, serão destacadas. Vá ao menu *File* e selecione *New>VCL Forms Application*. Adicione ao formulário principal os seguintes controles: um *Panel*, dois *Labels*, um *ListView*, quatro *Buttons*, um *StatusBar* e um *ProgressBar* (*Win32*). Ajuste-os conforme a **Figura 1**.

Dê um duplo clique sobre o *ListView* e adicione duas colunas. Altere a propriedade *Caption* da primeira para "Descrição" e da segunda para "URL". Modifique também a propriedade *ViewStyle* para *vsReport*.

Em seguida, clique duas vezes na *StatusBar*, inclua dois painéis, configure a propriedade *Width* do primeiro para 300 e a propriedade *Style* do segundo para *psOwnerDraw*. Agora, implemente o evento *OnDrawPanel* do *StatusBar* para o código da **Listagem 2**.

Isso posicionará a barra de progresso dentro do segundo

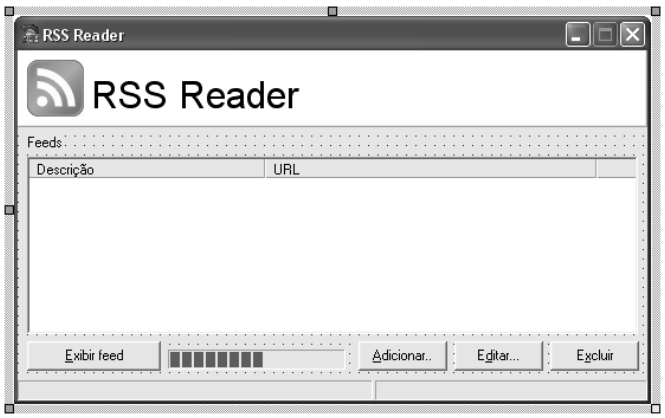


Figura 1. Organização do formulário principal

Listagem 1. Estrutura de um arquivo RSS baseado no XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.00">
  <channel>
    <title/>
    <link/>
    <description/>
    <language/>
    <copyright/>
    <image>
      <title/>
      <url/>
      <link/>
    </image>
    <item>
      <title/>
      <description/>
      <link/>
      <pubDate/>
      <guid/>
    </item>
  </channel>
</rss>
```

Listagem 2. Adicionando a barra de progresso ao StatusBar

```
procedure TfrmMain.StatusBarDrawPanel(
  StatusBar: TStatusBar; Panel: TStatusPanel;
  const Rect: TRect);
begin
  if Panel = StatusBar.Panels[1] then
  with ProgressBar1 do
  begin
    Top := Rect.Top;
    Left := Rect.Left;
    Width := Rect.Right - Rect.Left - 15;
    Height := Rect.Bottom - Rect.Top;
  end;
end;
```

painel da *StatusBar*. Porém, para esse código funcionar corretamente, precisamos configurar a propriedade *Parent* do *ProgressBar*. Então, adicione o evento *OnCreate* do formulário o código a seguir:

```
ProgressBar1.Parent := StatusBar1;
```

Para realizar um pequeno teste, altere a propriedade *Position* da barra de progresso para 50 e execute a aplicação (**Figura 2**).

Mude o nome do formulário para "frmMain" e salve a unit como "uFrmMain.pas". Salve também o projeto, com o nome "RSSReader.dpr". Vamos implementar os outros formulários da aplicação, voltando à janela principal mais tarde.

Criando o formulário de entrada de dados

Adicione um novo formulário ao projeto. Altere sua proprie-

dade *Name* para "frmAddFeed" e salve a unit como "uFrmAddFeed.pas". Essa janela será modal e, portanto, terá seu tempo de vida definido em runtime. Logo, precisamos retirá-la da lista de auto-criação do Delphi.

Para isso, vá ao menu *Project>Options* e selecione o item (ou aba, no Delphi 7) *Forms*. Em seguida, clique sobre o nome do formulário no *ListBox* da esquerda (*Auto-create forms*) e arraste-o para a lista da direita (*Availabe forms*).

Agora vamos montar a interface visual da janela. Adicione dois *Labels*, dois *Edits* e dois *Buttons*, organizando-os como mostrado na **Figura 3**.

Como o leitor pode ver na Figura, o primeiro *Edit* deverá se chamar "edtDescription" e o segundo, "edtURL". Por fim, configure as propriedades dos botões da seguinte forma:

- Botão OK: *Default*: *True*; *ModalResult*: *mrOK*;
- Botão Cancelar: *Cancel*: *True*; *ModalResult*: *mrCancel.*

Com isso terminamos esse formulário, que não terá uma linha de código sequer codificada.

Criando o formulário de visualização

Insira outro formulário ao projeto, dê o nome de "frmNews" e salve-o com o nome "uFrmNews.pas". Novamente, remova-o da lista de criação automática do Delphi. Adicione um *Panel* e um *ScrollBox* (*Additional*) e altere as propriedades para os valores a seguir:

• *Panel*: *Align*: *alTop*; *Caption*: "";

• *Scrollbox*: *Align*: *alClient*; *VertScrollBar.Smooth*: *True*; *VertScrollBar.Tracking*: *True*.

> **Nota:** *Smooth* ativa a rolagem "suave" da barra de rolagem vertical do *ScrollBox*, e *Tracking* faz com que o conteúdo do controle role durante o movimento da barra, e não apenas quando uma nova posição for definida.

Dentro do *Panel*, inclua dois *Labels* e um *Image* (*Additional*). Em seguida, mude o nome do primeiro *Label* para "lblChannelTitle", o segundo para "lblChannelDescription" e o *Image* para "imgChannelImage". Configure as propriedades desses controles da seguinte maneira:

• *lblChannelTitle*: *Font.Name*: Arial; *Font.Size*: 14; *Font.Style*: *fsBold*;

• *lblChannelDescription*: *Font.Name*: Arial; *Font.Size*: 10;

• *imgChannelImage*: *Align*: *alRight*; *Center*: *True*; *Transparent*: *True*;

Para organizá-los visualmente, oriente-se pela **Figura 4**.

Adicione um *Panel* dentro do *ScrollBox*. Chame-o de "pnlItem1" e configure sua propriedade *Align* para *alTop*. Dentro desse painel, adicione três *Label*, configurando-os como mostrado a seguir:

• Label 1: *Name*: "lblItemTitle"; *Align*: *alTop*; *Color*: *clInfoBk*; *Font.Name*: Arial; *Font.Size*: 11; *Font.Style*: *fsBold*;

• Label 2: *Name*: "lblPubDate"; *Align*: *alTop*; *Color*: *clGrayText*; *Font.Name*: Arial; *Font.Size*: 9; *Alignment: taRightJustify;*

• Label 3: *Name*: "lblItemDescription"; *Align*: *alClient*; *Color*: *clWindow*; *Font.Name*: Arial; *Font.Size*: 9;

> **Dica:** Utilize novamente a **Figura 4** para organizar visualmente os controles.

Com isso terminamos de desenhar a interface desse formulário. Vamos brincar com um pouco de código agora.

Baixando arquivos da Internet

Para ler o conteúdo do arquivo XML com as informações, precisamos realizar o download desse arquivo e processá-lo localmente. Existem muitas formas de fazer essa tarefa e vou apresentar aqui uma extremamente simples, sem precisar usar diretamente as API's da biblioteca *Wininet*.

Primeiramente, adicione a unit *ExtActns* na cláusula *uses* do

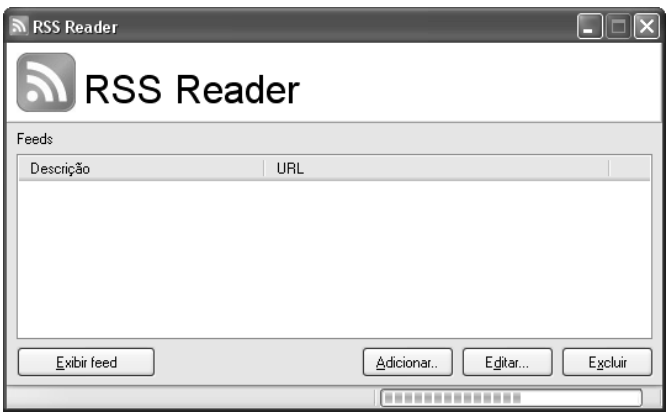


Figura 2. ProgressBar sendo exibida dentro da StatusBar

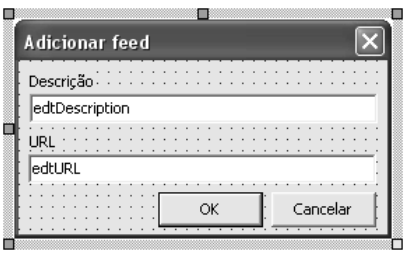


Figura 3. Formulário para adicionar feed

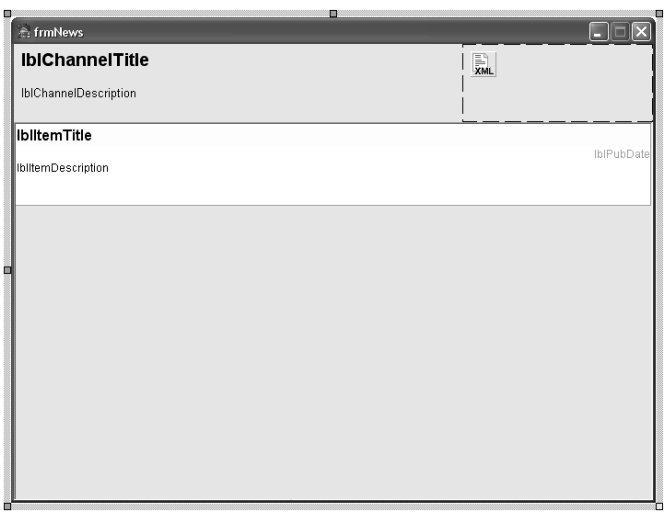


Figura 4. Formulário de visualização da notícias

uFrmNews. Crie uma nova função logo abaixo da declaração da classe do formulário, com a seguinte assinatura:

```
function DownloadFile(const AURL: string;
  AFileName: TFileName;
  ProgressProc: TDownloadProgressEvent = nil): Boolean;
```

Não podemos utilizar a combinação CTRL+SHIFT+C para implementar o código dessa função, pois não se trata de um método de uma classe, optamos por isso para poder usar a função na unit do formulário principal.

> **Nota:** Caso deseje, você pode implementar o código anterior e um unit separada.

Então, selecione o cabeçalho da rotina acima, cole na seção *implementation* da unit e codifique-o como mostrado na **Listagem 3**.

Listagem 3. Baixando arquivos com barra de progresso

```
function DownloadFile(const AURL: string;
  AFileName: TFileName;
  ProgressProc: TDownloadProgressEvent = nil): Boolean;
begin
  with TDownLoadURL.Create(nil) do
  try
    URL := AURL;
    FileName := AFileName;
    OnDownloadProgress := ProgressProc;
    try
      ExecuteTarget(nil);
      Result := True;
    except
      Result:=False;
    end;
  finally
    Free;
  end;
end;
```

A classe *TDownloadURL* é uma s*tandard action* (ação padrão), existente no componente *ActionList*. Ela permite realizar o download de um arquivo representado por uma URL, com acesso ao *feedback* do processo. A função que criamos apenas encapsula o código necessário para isso, com o objetivo de facilitar a utilização da classe.

> **Dica:** Consulte a edição 72 da revista ClubeDelphi para saber mais sobre s*tandard action*s.

Com mais um passo concluído, vamos agora à parte mais importante, e interessante, desse aplicativo: processar o XML com as notícias e exibí-las ao usuário.

Exibindo as notícias baixadas

Para podermos mostrar o conteúdo do RSS visualmente, precisamos realizar o *parser* do arquivo XML correspondente. O Delphi oferece mais de uma maneira de conseguir isso, cada uma com suas vantagens e desvantagens. Como a formatação do XML varia em função da versão do RSS em questão, vamos fazer uso do *XMLDocument* (paleta *Internet*), que facilita a leitura de arquivos XML com estruturas dinâmicas.

Então, adicione-o ao *frmNews* e deixe suas propriedades como estão. Volte ao código e crie o método a seguir, na seção *public* do formulário:

```
procedure LoadFromXML(const AFileName: TFileName);
```

Pressione CTRL+SHIFT+C para implementar o método e digite o código da **Listagem 4** (os números na margem esquerda existem apenas para indicação).

O código está comentado, mas vamos explicar alguns pontos (destacados) menos triviais:

1. A função auxiliar *GetValue* serve para facilitar a leitura de uma determinada tag dentro do XML. Como é preciso checar se a marcação existe, seria custoso ter verificações iguais em vários pontos diferentes do método. *GetValue*, então, encapsula esse comportamento, retornando uma *string* vazia caso a tag não seja encontrada no XML. Outra tarefa realizada é a remoção de caracteres de quebra de linha, que podem distorcer a representação visual dos *Labels*;

2. A propriedade *DocumentElement* da classe *TXMLDocument* acessa a tag raiz do XML. No nosso caso, a marcação *<rss>*. A partir daí podemos navegar hierarquicamente pelo resto do arquivo. Essa propriedade é do tipo *IXMLNode*, que representa um nó dentro da estrutura do documento. Pesquise no help do Delphi para conhecer mais sobre essa interface;

3. Para localizar uma tag dentro do documento, utilizamos o *FindNode*, da propriedade *ChildNodes*. Caso a tag exista, será retornada uma interface *IXMLNode*, mencionada no item anterior, e do contrário, *nil*;

4. Aqui estamos usando a função criada anteriormente, para realizar o download de um arquivo. Nesse caso, trata-se da imagem do provedor do *feed* RSS;

5. A maioria das imagens disponibilizadas são no formato JPEG ou GIF. O Delphi suporta o primeiro nativamente, mas não o segundo. Para podermos exibir imagens GIF, precisamos de um componente de terceiros. No nosso exemplo, usaremos a JVCL, que fornece a unit *JvGif*. Adicione-a à cláusula *uses* a unit e a classe *TPicture* ganhará a capacidade de exibir vários formatos de imagem;

> **Nota:** Alguns sites fornecem suas imagens no formato PNG. Tanto o Delphi quanto a JVCL **não** possuem suporte nativo para esse tipo de arquivo.

6. A maneira mais simples de exibir as notícias contidas no arquivo XML seria usando o tradicional *layout* tabular, com Grids etc. Porém, no caso do RSS, isso pode comprometer a visualização do conteúdo do *feed*. Então, para tornar mais confortável a leitura por parte do usuário, estamos mostrando cada item dentro de um painel, com seus atributos distribuídos por *Labels*. Como não há como prever quantos painéis podemos precisar, vamos criá-los em runtime, e isso envolveria criar cada painel separadamente, os controles internos e configurar as propriedades e eventos de cada um. Para poupar todo esse trabalho, escrevi a função *CloneControl*, que, como o nome sugere, faz cópias de um controle qualquer, descendente de *TControl*, e de todos os controles que são seus "filhos". Como isso funciona? Vamos descobrir no próximo tópico;

7. Para fazer referência aos controles clonados dentro do painel, usamos o *FindComponent*. A função *CloneControl* irá nomeá-los com o nome do pai + o nome do controle.

Clonando controles

O item 6 descreve bem uma situação onde clonar controles pode economizar tempo e trabalho. O código não está publicado aqui por razões de espaço, mas *CloneControl* faz parte da unit *CloneTools*, disponível para download juntamente com o exemplo deste artigo. Nesse arquivo existem, além da rotina mencionada, mais algumas para clonar formulários, propriedades e eventos.

A "mágica" é a mesma usada pelo IDE do Delphi para salvar e restaurar os atributos dos componentes criados em *design time* para o arquivo do formulário (DFM): criar uma nova instância de objeto, do mesmo tipo do artefato sendo clonado, serializar as propriedades do primeiro e restaurá-las no segundo.

Os eventos precisam de um código extra para controles não descendentes de *TCustomForm*, pois esse último possui um comportamento diferente dos demais. Por fim, a função verifica se o objeto sendo clonado é derivado de *TWinControl*, o que indica que ele pode, teoricamente, conter outros controles, e, em caso positivo, chama a si própria para clonar cada componente filho encontrado.

> **Dica:** Não deixe de consultar o código da unit *CloneTools* para entender como o mecanismo funciona.

A função *CloneControl* recebe três parâmetros, como mostrado no código a seguir:

```
function CloneControl(Control: TControl;
  NewName: TComponentName;
  NewParent: TWinControl = nil): TControl;
```

O primeiro argumento é o controle sendo clonado, o segundo, o nome que ele possuirá (isso é importante, pois não podem existir dois componentes com a mesma propriedade *Name*) e o terceiro é o novo *Parent* que a nova instância terá.

Se não for informado, será assumido o mesmo pai do objeto sendo copiado. Por fim, a função retorna um ponteiro para o controle recém-criado. É importante frisar que o *Owner* desse componente será o mesmo do seu "irmão gêmeo". Portanto, a responsabilidade de destruí-lo posteriormente será também desse proprietário.

Entendido como funciona a clonagem de controles, prossigamos com o desenvolvimento do nosso leitor RSS.

De volta ao formulário principal

Vamos finalizar a implementação do formulário principal. Adicione a unit *frmAddFeed* à cláusula *uses* (ALT+F11), e implemente o evento *OnClick* dos botões *Adicionar*, *Editar* e *Excluir* com o código mostrado na **Listagem 5**.

Listagem 4. Processando o XML e exibindo os dados visualmente

```
procedure TfrmNews.LoadFromXML(const AFileName: TFileName);
{ Função auxiliar para extrair dados das tags XML }
1 function GetValue(NodeName: WideString;
  NodeList: IXMLNodeList): WideString;
  var
    Node: IXMLNode;
  begin
    Result := '';
    Node := NodeList.FindNode(NodeName);
    if Assigned(Node) then
    begin
      Result := AnsiReplaceText(Node.Text, #10, '');
      Result := AnsiReplaceText(Result, #13, '');
    end;
  end;
var
  RSSVersion: string;
  ChannelNode, ImageNode, ItemNode: IXMLNode;
  Panel: TPanel;
  I, ATop: Integer;
  PanelName: TComponentName;
  ImageFileName: TFileName;
  ImageURL: string;
  ChildLabel: TLabel;
begin
  { Carrega o arquivo XML para o XMLDocument }
  XMLDocument1.FileName := AFileName;
  XMLDocument1.Active := True;
  try
    { Verifica a versão do RSS }
2   RSSVersion := XMLDocument1.DocumentElement.Attributes['version'];
    if RSSVersion[1] <> '2' then
      raise Exception.Create('Apenas a versão 2.0 do formato RSS é ' +
        'suportada por este aplicativo');
    { Pega os dados do canal no arquivo }
3   ChannelNode := XMLDocument1.DocumentElement.
      ChildNodes.FindNode('channel');
    if Assigned(ChannelNode) then
      with ChannelNode do
      begin
        lblChannelTitle.Caption := GetValue('title', ChildNodes);
        lblChannelDescription.Caption := GetValue(
          'description', ChildNodes);
      end;
    { Pega os dados da imagem do produtor do RSS }
    ImageNode := ChannelNode.ChildNodes.FindNode('image');
    if Assigned(ImageNode) then
      with ImageNode do
      begin
        imgChannelImage.Hint := 'Ir para ' + GetValue('title', ChildNodes);
        ImageURL := GetValue('url', ChildNodes);
      end;
    { Baixa a imagem para o disco }
    if ImageURL <> '' then
    begin
      ImageFileName := ExtractFilePath(AFileName) +
        'image' + ExtractFileExt(ImageURL);
4     if DownloadFile(ImageURL, ImageFileName) then
5       imgChannelImage.Picture.LoadFromFile(ImageFileName);
    end;
    { Percorre o arquivo carregando os itens de notícia }
    I := 0;
    ATop := 0;
    ItemNode := ChannelNode.ChildNodes.FindNode('item');
    repeat
      if Assigned(ItemNode) then
      begin
        Inc(I);
        { Clona o painel criado em design time }
        PanelName := Format('pnlItem%d', [I + 1]);
6       Panel := CloneControl(pnlItem1, PanelName) as TPanel;

        Panel.Top := ATop;
        Panel.Align := alTop;
        Panel.Visible := True;
       { Importante somar um pixel para o alinhemento automático funcionar }
        Inc(ATop, Panel.Height + 1);
        { Atualiza os labels internos dos painéis }
7       ChildLabel := FindComponent(PanelName + 'lblItemTitle') as TLabel;
        if Assigned(ChildLabel) then
        begin
          ChildLabel.Caption := GetValue('title', ItemNode.ChildNodes);
          ChildLabel.Hint := GetValue('link', ItemNode.ChildNodes);
        end;
       ChildLabel := FindComponent(PanelName + lblItemDescription') as TLabel;
        if Assigned(ChildLabel) then
        begin
          ChildLabel.Caption := GetValue('description', ItemNode.ChildNodes);
          ChildLabel.Hint := GetValue('link', ItemNode.ChildNodes);
        end;
        ChildLabel := FindComponent(PanelName +
          'lblPubDate') as TLabel;
        if Assigned(ChildLabel) then
          ChildLabel.Caption := GetValue('pubDate', ItemNode.ChildNodes);
      end;
      { Pega a próxima tag XML do arquivo }
      ItemNode := ItemNode.NextSibling;
    until not Assigned(ItemNode);
    Self.Caption := Format('%s - %d notícia(s)', [Caption, I]);
  finally
    XMLDocument1.Active := False;
  end;
end;
```

O código é bem simples: no primeiro exibimos o formulário de adição para incluir um novo *feed* no *ListView*. No segundo, carregamos o mesmo formulário com os dados do item selecionado na lista, e atualizamos de volta caso necessário. Por fim, no terceiro excluímos o elemento selecionado do *ListView* através do *Delete*.

O último botão a ser codificado é o botão que exibe as notícias do site selecionado. Como mencionamos anteriormente, precisamos copiar o arquivo XML do servidor Web onde ele está armazenado para a máquina local. Então, dê um duplo clique sobre o botão e digite o código da **Listagem 6**.

O código é auto-explicativo. A unica coisa que precisa ser dita é como criar o *callback* para a função *DownloadFile*, a fim de fornecer um feedback visual ao usuário sobre o processo de download.

Para conhecer a assinatura correta, basta pesquisar no help da classe *TDownloadURL* pelo evento *OnProgress*. Então, declare um novo método na seção *public* do formulário, como mostrado na **Listagem 7**.

Implemente-o pressionando CTRL+SHIFT+C com o código da **Listagem 8**.

Muito bem. Já temos o suficiente para compilar e executar a aplicação. Então, pressione F9, clique no botão *Adicionar* e insira a URL de algum feed RSS (**Figura 5**).

No nosso exemplo usaremos os seguintes links:

```
http://z.about.com/6/g/delphi/b/rss2.xml
http://info.abril.com.br/aberto/infonews/rssnews.xml
http://oglobo.globo.com/rss/plantaotecnologia.xml
http://rss.terra.com.br/0,,EI1977,00.xml
http://rss.noticias.uol.com.br/ultnot/index.xml
```

Selecione algum item no *ListView* e clique no *Exibir feed*. Veja o resultado na **Figura 6**.

> **Nota:** Se você tiver um *firewall* ativado, provavelmente ele bloqueará o acesso à Internet da nossa aplicação.

> **Nota:** O feed do site O Globo possuía, na data da produção desse artigo, uma imagem no formato PNG. Como dito anteriormente, será levantado uma exceção ao se tentar carregar a figura indicada no XML. O link foi listado acima propositalmente, para frisar que, para este formato específico de imagem, deve-se escrever um código de tratamento especial, ou usar um componente de terceiros (seção *Links*)

Dandos os retoques finais

Para terminarmos nosso leitor de RSS, vamos dar um aspecto de *links* para as notícias e abrir o browser padrão do sistema, quando elas forem clicadas. Então, volte ao *frmNews* e implemente os eventos *OnMouseEnter*, *OnMouseLeave* e *OnClick* do *lblItemTitle* com o código da **Listagem 9**.

> **Nota:** Os eventos OnMouseEnter e OnMouseLeave existem apenas do Delphi 2005 em diante. Para realizar isso em versões anteriores, é preciso lidar diretamente com mensagens do Windows. Se você tiver dificuldades em realizar essa tarefa, mande um email para mim.

Listagem 5. Adicionando um novo feed

```
procedure TfrmMain.btnAddFeedClick(Sender: TObject);
begin
  with TfrmAddFeed.Create(nil) do
  try
    if ShowModal = mrOK then
      with lstFeeds.Items.Add do
      begin
        Caption := edtDescription.Text;
        SubItems.Add(edtURL.Text);
      end;
  finally
    Release;
  end;
end;

procedure TfrmMain.btnEditFeedClick(Sender: TObject);
begin
  with TfrmAddFeed.Create(nil) do
  try
    with lstFeeds.Selected do
    begin
      edtDescription.Text := Caption;
      edtURL.Text := SubItems[0];
      if ShowModal = mrOK then
      begin
        Caption := edtDescription.Text;
        SubItems[0] := edtURL.Text;
      end;
    end;
  finally
    Release;
  end;
end;

procedure TfrmMain.btnDeleteFeedClick(Sender: TObject);
begin
  lstFeeds.Selected.Delete;
end;
```

Listagem 6. Código do botão Exibir feed

```
uses uFrmNews;

procedure TfrmMain.btnShowNewsClick(Sender: TObject);
var
  LocalFileName: TFileName;
begin
  { O nome do arquivo XML local será "headline" +
    O índice do feed selecionado no ListView }
  LocalFileName := Format('headlines%d.xml',
    [lstFeeds.Selected.Index]);
  if DownloadFile(lstFeeds.Selected.SubItems[0],
    LocalFileName, DownloadProgress) then
  begin
    Beep;
  end
  else
    raise Exception.Create(
      'Não foi possível realizar o download do ' +
      'arquivo XML relativo ao feed selecionado');
  with TfrmNews.Create(Application) do
  begin
    Caption := lstFeeds.Selected.Caption;
    LoadFromXML(LocalFileName);
    Show;
  end;
end;
```

Listagem 7. Callback Download Progress

```
uses ExtActns;
(...)
public
  procedure DownloadProgress(Sender: TDownLoadURL;
    Progress, ProgressMax: Cardinal;
    StatusCode: TURLDownloadStatus;
    StatusText: string; var Cancel: Boolean);
end;
```

Listagem 8. Implementando o método de callback

```
procedure TfrmMain.DownloadProgress(
  Sender: TDownLoadURL; Progress,
  ProgressMax: Cardinal;
  StatusCode: TURLDownloadStatus; StatusText: string;
  var Cancel: Boolean);
begin
  ProgressBar1.Max := ProgressMax;
  ProgressBar1.Position := Progress;
  StatusBar1.Panels[0].Text := StatusText;
end;
```

Listagem 9. Transformando labels em links

```
procedure TfrmNews.lblItemTitleMouseEnter(Sender: TObject);
begin
  (Sender as TLabel).Font.Color := clBlue;
end;
procedure TfrmNews.lblItemTitleMouseLeave(Sender: TObject);
begin
  (Sender as TLabel).Font.Color := clBlack;
end;
procedure TfrmNews.lblItemTitleClick(Sender: TObject);
begin
  ShellExecute(Handle, 'open', PAnsiChar((
    Sender as TLabel).Hint), nil, nil, SW_SHOWNORMAL);
end;
```

> **Nota:** Adicione a unit *ShellAPI* à cláusula *uses* do arquivo para dar suporte à *ShellExecute*.

Por fim, resta trocar o cursor dos *Labels* pelo ícone da "mãozinha" clássica para denotar um link. O Delphi possui o cursor *crHandPoint*, que pode ser usado nesse caso. Porém, vamos usar o cursor padrão do Windows, a mesma "mãozinha" do Internet Explorer. Então, implemente o evento *OnCreate* do formulário e digite o código a seguir:

```
Screen.Cursors[NIDC_HAND] := LoadCursor(0, IDC_HAND);
lblItemTitle.Cursor := NIDC_HAND;
lblItemDescription.Cursor := NIDC_HAND;
```

Estamos carregando o cursor direto dos recursos do sistema operacional. Por fim, declare na seção *const* da unit o seguinte código:

```
NIDC_HAND = 32649;
```

Como vamos clonar o painel e seus controles em tempo de execução, esse cursor também será configurado para as respectivas cópias. Salve tudo e execute a aplicação. Veja o resultado final na **Figura 7**.

Conclusão

Vimos nesse artigo como criar nosso próprio leitor de notícias RSS. Como você pôde notar, tudo gira em torno de processar um arquivo XML. Aprendemos também como clonar controles em tempo de execução, dando uma aparência mais profissional para o aplicativo, sem gastar tempo codificando.

Além de construir um leitor independente de notícias, você pode implementar a mesma idéia mostrada aqui em seus softwares, oferecendo, por exemplo, notícias para seus clientes sobre novas versões, correção de bugs, promoções etc., tudo de dentro da sua aplicação.

Um forte abraço a todos e até a próxima! ■

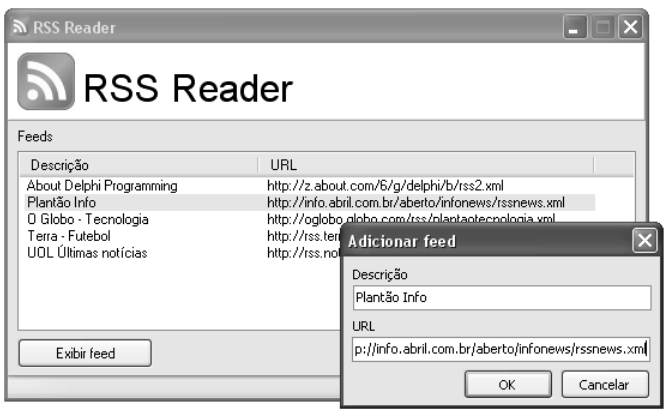


Figura 5. Inserindo um feed RSS

Links

Wiki sobre o formato RSS

en.wikipedia.org/wiki/Rss_(file_format)

Wiki sobre sindicalização na Web

en.wikipedia.org/wiki/Web_syndication

Especificação do RSS

www.rssboard.org/rss-specification

Componente para exibição de imagens PNG

http://pngdelphi.sourceforge.net/

Figura 6. Notícias sendo exibidas pela aplicação

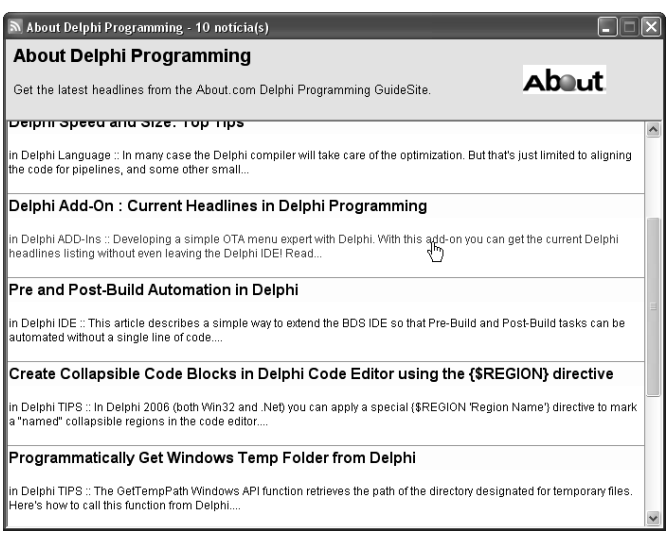


Figura 7. Efeito MouseOver e mãozinha do Windows

Complete sua coleção!

Feita para Desenvol...

SQL magazine

Escolha 03 edições e pague **R$ 6,90** em cada revista.

0

apeamento Objeto acional. Aprenda a construir mapeador O/R em .NET.

arehouse Builder. Construa a data warehouse com a amenta ETL da Oracle-tel

ML. Aprenda passo a passo no construir um diagrama de uências.

imização. Influenciando o nizador de consultas.

21

- Normalização. Aprenda a organizar melhor o esquema de seu banco de dados

- PostreSQL. Saiba o que são rules e como utilizá-las.

- SQL Server 2000. Entenda os modelos de recovery.

- InterBase 7.5. Quando e porque migrar.

- Particionamento do Oracle. Perigos e armadilhas Parte II.

22

- Backup. Saiba como proteger seus dados no Oracle e no SQL Server.

- UML. Conheça algumas opções de ferramentas livres - Parte II.

- Homologação de dados. Saiba como avaliar a qualidade de seu modelo de dados.

- Especial. Acessando servidores Oracle a partir do SQL Server 2000.

23

- Proteja-se. Saiba como funciona o SQL Injection e aprenda a lidar com ele no SQL Server.

- Administrando o Oracle. Conheça o Oracle Enterprise Manager.

- Backup. Saiba como efetuar o restore de Bds no SQL Server.

- Modelagem. Utilizando padrões na elaboração do projeto de banco de dados.

24

- Java + BD + Web. Trabalh forma integrada com: MySQ Tomcat e Java com NetBea

- Oracle 10g. Aumente a disponibilidade de seu SGB com o automatic Storage Management.

- PostgreSQL. Veja na práti como efetuar replicação ent servidores PostgreSQL.

ustering no SQL Server. Saiba a que serve e como funciona a tecnologia de redundância.

Expert. Configure e administre e de dados no Firebird e rbase.

aché. trabalhe com objetos tamente no banco de dados.

racle Discoverer. A prenda so a passo a trabalhar com a amenta de análise de dados da cle.

26

- Mineração de dados. Entenda como ocorre a descoberta de informações em base de dados.

- IBExpert. Administrando o InterBase/Firebird. Conheça algumas características avançadas.

- UML. Ferramenta de modelagem free com engenharia reversa.

- MySQL. Transforme-se em um especialista na otimização de consultas - Parte II.

27

- MySQL. Veja como rodar um banco de dados MySQL em várias máquinas ao mesmo tempo!

- ER/Studio. Conheça esta excelente ferramenta para modelagem de dados.

- SQL. Descobrindo associações entre pares de linhas.

- SQL Server. Veja como enviar e-mails anexos no SQL Server sem o uso do SQL Mail.

28

- Dúvidas frequentes em bancos de dados.

- Desafio SQL. Construindo um modelo para um jogo de futebol on-line.

- Index Tuning Wizard. Como produzir tunings automáticos.

- Erwin conheça esta ferramenta de modelagem E/R.

29

- Oracle FLASHBACK. Recupere informações para u determinado ponto do tempo

- SQL Server. Descubra se vo esta trabalhando corretamen com o SQL Server.

- InterBase / Firebird. Trabalhando com stored procedures e triggers.

www.devmedia.com.br/msdn/anteriores

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700